Simplex^{TM} in a Hostile Communications Environment: The Coordinated Prototype

Neal Altman
Chuck Weinstock
Lui Sha
Danbing Seto

August 1999

TECHNICAL REPORT
CMU/SEI-99-TR-016
ESC-TR-99-016

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **AUG 1999**	2. REPORT TYPE **N/A**	3. DATES COVERED -

4. TITLE AND SUBTITLE **SimplexTM in a Hostile Communications Environment: The Coordinated Prototype**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University Libraries 5000 Forbes Avenue Pittsburgh, PA 15213-3890**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**81**	

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Simplex^{TM} in a Hostile Communications Environment: The Coordinated Prototype

CMU/SEI-99-TR-016
ESC-TR-99-016

Neal Altman
Chuck Weinstock
Lui Sha
Danbing Seto

August 1999

Dependable Systems Upgrade

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L Compton

Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

Acknowledgements

The authors gratefully acknowledge the contribution of the Simplex team members for the completion of this report. Mike Gagliardi provided significant input, particularly with respect to the software architecture, design evolution and lessons learned. Ted Marz was instrumental in detailing the communications and attack software design and implementation as well as providing insightful and challenging reviews of the document as a whole. John Walker clarified many questions with respect to the system hardware development and configuration as well as giving several helpful reviews of the document. Peter Feiler and Dave Gluch provided timely information on their use of Coordinated Prototype in their own work.

Abstract

Simplex is an engineering framework embodying fault tolerance and dynamic upgrade in a highly maintainable system. This report describes an approach to using Simplex to construct a COTS-based computer system capable of coordinated real-time motion control in a hostile communications environment. It also discusses how selected portions of the Coordinated Prototype tolerate software faults and allow new software to be deployed and tested during system operation.

1 Problem Statement

> *Accelerating rates of change will make the future environment more unpredictable and less stable, presenting our Armed Forces with a wide range of plausible futures. Whatever direction global change ultimately takes, it will affect how we think about and conduct joint and multinational operations in the 21st century. How we respond to dynamic changes concerning potential adversaries, technological advances and their implications, and the emerging importance for information superiority will dramatically impact how well our Armed Forces can perform its duties in 2010. [DoD ND]*

Superior intelligence and the technology for its effective application provide a force multiplier of proven value. Timely collection, distribution, and interpretation of information is already an important component of military systems and it is reasonable to assume that its value will increase in the immediate future. As information becomes more abundant and timely, cooperating force components can make increasing use of real-time information to coordinate their activities and respond to changes in the operating environment.

While the increased use of off-platform sensors and real-time processing allows flexible response to rapidly changing situations, increasing dependence on information delivered during missions requires mitigation of risks as well as the exploitation of benefits. If communications are interrupted, how can systems compensate? When the form and content of information changes as new sensors and information sources are fielded, how can existing systems be upgraded to adapt?

To explore these issues, we adapted Simplex^{TM} technology to a communications dependent environment and developed the Coordinated Prototype. In this prototype, two independent computer systems attempt to perform a task that requires coordination using message exchange to synchronize their actions. The communications link between the systems is subject to degradation or interruption during operation and the system must compensate for missing information. The software is also tolerant of software faults, maintaining safe operation in the face of errors in active software.

To provide clear, visible evidence of success or failure, the prototype uses the problem of coordinating the movement of inverted pendulums. Each pair of inverted pendulums may move independently under computer control. Through communications links, a separate system can provide a destination for each pendulum. The pendulums then advance to the target location.

The focus of the prototype demonstration is to maintain coordinated control of the pendulum pair. That is, they must advance in synchronization. If the pendulum motion is severely out of synchronization the pendulums will fall. In demonstrations, faulty control software upgrades and disruptions to external communication are introduced without causing a light rod connecting the pendulum tips to drop.

The Coordinated Prototype addresses the following questions by applying Simplex technology:

- Can coordination be maintained in the face of communications problems?
  - What are the effects of partially or completely denying communications for a period?
  - What are the effects of degraded communications, where communications links are overloaded with spurious messages?
- Can platform safety be assured when software is altered?
  - What if the system is upgraded with partially tested software prior to mission start?
  - What if software is changed during operation?

The Coordinated Prototype design was not intended to represent any specific system or replicate a particular mission profile.

The next section describes the concept of the prototype—how we model the communications problem using dual inverted pendulums. The third section introduces Simplex concepts, and gives an overview of how these concepts are applied to the dual pendulum prototype. For those interested, Section Four provides implementation details, followed by sections on demonstration scenarios and analytical studies. In the final section, we conclude with a discussion of lessons learned.

2 A Prototype for Coordinated Control

Consider a single aircraft or missile. In itself, an air vehicle typically contains the necessary sensors and controls to maneuver from point to point in an autonomous fashion. However, a single vehicle is frequently deployed as part of a larger mission, moving in concert with other elements. Group action can be achieved in various ways. With sufficient pre-planning, individual vehicles can achieve coordination by rigidly adhering to a schedule. If nothing changes, individual vehicles can be partially or completely unaware of the other participants, simply conforming to the schedule. In early air missions, group action was achieved by flying in formations that coordinated by maintaining visual contact. As off platform sensors and effective communications became available, aircraft were more able to react to events that were beyond the sensor range of the air vehicle. All of these operational modes are still relevant, but the possibility for flexible response based on remote input is the most challenging scenario and was considered the most interesting problem for the Coordinated Prototype.

As a further example, consider midair refueling—a widely employed technique for extending military aircraft range and payload. Since the refueling aircraft are generally vulnerable and usually of quite different types than the refueled aircraft, they seldom travel together throughout a mission. At its simplest, two aircraft must rendezvous and then fly in close formation to transfer fuel. With sufficient preparation, this rendezvous can be achieved without radio communication, providing security advantages. However, pre-planned arrangements may not be sufficient due to the variability inherent in military flying. Plans must be altered in that case to bring available tankers and aircraft together within a limited time frame.

A general design for the prototype was derived from the above observations along with the desire to simulate the problem of coordinating such systems in a hostile communications environment. Inverted pendulums were used in the design to provide visible evidence of a communications failure. Previous Simplex prototypes were based on inverted pendulums and therefore, they provided a fairly low risk way of providing a meaningful—though abstract—test environment. The decision was made to use as much COTS hardware and software as possible.

An inverted pendulum is an intrinsically unstable device, which requires continuous real-time control to stay upright. In addition to keeping the pendulum upright, it is possible to write controllers that can move the pendulum to arbitrary locations on the track and statically balance the pendulum in a vertical position. These functions can be defined in combination, giving a system that will attempt to balance the pendulum at a specified location on the track. These movement combinations were demonstrated in earlier Simplex prototypes.

Although a very modest personal computer or equivalent can easily provide control for an inverted pendulum, multiple computers have been used in some Simplex prototypes to demonstrate how hardware redundancy is compatible with the Simplex engineering framework. In these prototypes, a COTS network was successfully used to handle the distribution of real-time pendulum control data. These configurations were the basis for the Coordinated Prototype.

The primary scenario for the Coordinated Prototype is to bring the two pendulums into synchronized movement and move them together to specified locations on the track (coordinated motion). At any time, communications may be partially or completely blocked between the two pendulums. Even when communications are blocked, the pendulums must stay together. This functionality requires additional control algorithms for coordination, as well as predictable movement algorithms to hold the pendulums together.

For coordinated motion, pendulum controllers must hold the pendulum within a “box” centered on the current position of the other pendulum. In addition, as target locations are entered and changed, the pendulums must accelerate and decelerate together to reach the goal. Two strategies can achieve this result. The first is to use controllers whose behavior is highly predictable so that a synchronized start signal is sufficient to keep the pendulums together. The second is to exchange position information frequently so the controller is constantly aware of the other pendulum’s position. The first strategy is highly autonomous, but requires great precision and predictability. The second has demanding communications requirements, but the controller need not be very precise.

In practice, a mixed strategy was developed. The pendulum controllers are relatively precise; the pendulum can be kept upright without large excursions from a desired position and the direction and motion rate of the pendulum can be varied predictably. The position of the other pendulum is known through regular updates. In case of communications interruption, the pendulums reduce speed and then stop as the position updates became less frequent.

The pendulum controller moves the pendulum to achieve several goals: small movements to keep the pendulum upright, acceleration or deceleration towards a desired target location and velocity adjustment (or station keeping) to stay in synchronization with the other pendulum. A single controller need not accomplish all of these tasks; in the Coordinated Prototype, simple pendulum motion is handled by an inner control loop which performs movement to a specified goal position (similar to the controller for previous Simplex prototypes) and synchronized movement by an outer coordination loop which coordinates motion between the two pendulums by exchanging pendulum position information and feeding target locations to the inner controllers.

There were two advantages to this design. First, most of the algorithms for the inner control loop were available from previous prototypes, allowing us to concentrate on developing the strategies for the outer coordination loop. Second, the two loops could be executed at different rates, with the speeds matching their different timing requirements, consuming fewer system resources. The ability to adjust execution rates independently proved useful in resolving communications bottlenecks.

3 Simplex Concepts

As illustrated in the previous section, just as it is relatively easy to coordinate aircraft on a mission in a benign environment, it is also relatively easy, given the right algorithms, to coordinate twin pendulums. What happens, however, if the environment is not so benign? Can we continue to coordinate our aircraft or pendulums in a hostile environment? How does our prototype system have to be enhanced to do this successfully? Finally, if we want to introduce improvements to the prototype (that is, upgrade it) can we do so in a safe and reliable manner? In this section, we discuss these questions.

In order to answer the above questions affirmatively, our system needs to exhibit the following characteristics:

- The individual pendulum controllers must be able to ensure that their pendulum remains upright at all times.
- Each of the individual pendulum controllers must be able to deal with the partial or complete loss of communications with the other pendulum controller and do the "reasonable" thing.
- It must be possible to introduce new software into the system while ensuring that both of the above conditions are still met.

The Simplex Architectural Framework, along with some careful system design, allows us to achieve these goals. The major components of this framework are Dynamic Binding, Analytic Redundancy and Rate Monotonic Scheduling (RMS) Theory.

Dynamic binding is the mechanism that allows new software components to be inserted into the running system. The combination of Analytic Redundancy and RMS Theory are used to guarantee that the controller is able to keep the pendulum upright at all times, to deal with the temporary loss of communications between pendulum controllers, and to deal with the insertion of possibly "buggy" new software. We will discuss each of these components in turn.

This section will also discuss the issues related to the use of Simplex and COTS software including their application to the Coordinated Prototype and will close with an overview of the benefits of Simplex.

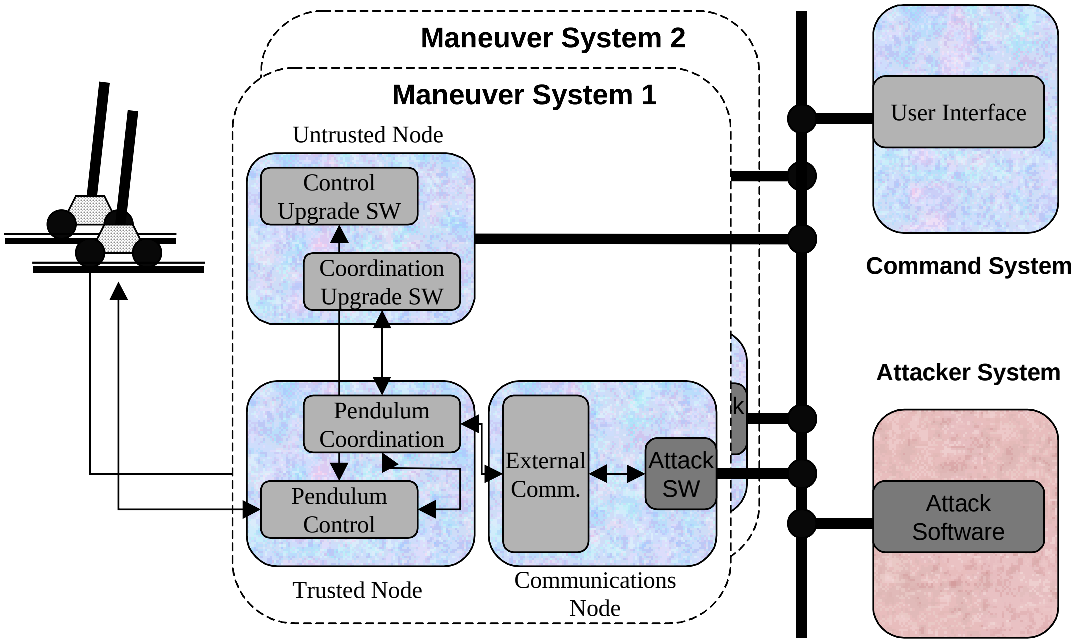


Figure 1: Coordinated Prototype Sofware Architecture Block Diagram

3.1 Dynamic Component Binding

Dynamic binding is the set of features that permits the creation and deletion of a portion of a system during execution. Using dynamic binding, portions of a system can be altered without stopping the entire system facilitating fault resistance and software upgrades. Because interruption of service is not acceptable during the execution of a real-time program, dynamic component binding must be performed without stopping the flow and processing of information although the software creating or processing information flows may be varied. The specific communication technology that is used by Simplex is a publish/subscribe mechanism as described in [Rajkumar 95]. The producer of a piece of data “publishes” it. Any subsystem that wishes to make use of the data “subscribes” to it. The implementation is based on POSIX .1b message queues and is similar to the labeled message construct in POSIX .21.

The Coordinated Prototype uses dynamic binding to allow the seamless insertion of upgraded control algorithms into the system while maintaining continuous real-time control of the pendulums.

3.1.1 Replacement Units

Dynamic components are packaged as **replacement units**. A replacement unit is a process abstraction which encapsulates a user-defined set of application functionality with a template for communicating with other replacement units and sufficient address space protection such that any one replacement unit cannot corrupt the address space of any other replacement unit.

Replacement units are designed in such a way that they are able to replace an existing replacement unit or be replaced by a new replacement unit without interrupting system operation. Replacement units allow dynamic upgrade, through replacement. With fault detection and functional replication, replacement units allow fault recovery by implementing analytical redundancy.

The Simplex Engineering Framework allows any type functionality to be encapsulated in a replacement unit. In the Coordinated Prototype, replacement units are used to hold some (but not all) of the real-time controllers and coordinators. In the prototype, three real-time controller slots were provided for the inner control loop and three slots for the outer coordination loop. A slot may be encapsulated as a replacement unit or a controller or coordinator can be "hard-wired" into the slot. In the prototype, a single controller in each control and coordination loop was packaged as a replacement unit ("Upgrade SW" in Figure 1 on page 8) and the remaining slots contain hard-wired controllers (see also Figure 15 on page 45). Controller slots and the controllers or coordinators they contain are named, in order of increasing stability (and decreasing sophistication): upgrade, baseline, and safety. This ordering becomes important in fault correction. For instance, the safety controller serves as the last chance control for the system, while the baseline controller is preferred over the safety controller for normal operation.

3.1.2 Fault Containment

Replacement units encapsulate code to perform a specific function and allow the code to be replaced with new code as needed. In addition to allowing software upgrade, replacement units facilitate fault containment. Fault containment is the process of preventing flaws in a form propagating to other parts of a system. Fault containment, in itself, neither prevents failure nor repairs the failure. Fault containment does help assure the success of fault recovery when redundancy exists.

Application faults can cause damage in several ways and the prototype uses several strategies to contain faults. Process address space protection and interprocess group communication are used to contain runtime errors. Rate Monotonic Analysis (RMA) and enforcement of deadlines contain timing faults. Analytic redundancy provides protection from semantic faults by using multiple controllers whose output is dissimilar and trading off between the reliability and the performance of these controllers. Finally, the prototype isolates untrusted replacement units on separate nodes, where the potentially unreliable combination of deliberately malicious software, COTS hardware and COTS software can be isolated.

3.2 Analytic Redundancy

Redundancy, the provision of backup components in case of failure, is a common technique in fault tolerant systems. Analytic redundancy is a specific kind of redundancy, which is best illustrated by example.

Suppose there are two subsystems that accomplish the same task. One of them is preferred because it produces “better” results. It may be deemed better for any number of reasons. Perhaps it produces more accurate values, or it produces the values faster, or perhaps it uses fewer resources. For whatever the reason, we prefer to use the results of this subsystem. However, in the event that the preferred subsystem fails to accomplish its task, we are happy to use the “inferior” results of the non-preferred subsystem. The preferred and the non-preferred systems are said to be analytic redundant because they accomplish the same task, though by different methods and with different semantic results.

Many systems exhibit analytic redundancy. For instance the power-assisted steering in most vehicles is an analytically redundant system. We prefer the power assist because it makes steering easier. But if the power assist is lost (e.g., due to an engine failure or a belt breakage), we can fall back upon the mechanical steering.

Simplex uses analytic redundancy as a key enabling technology. Instead of simply creating multiple copies of a component considered likely to fail, the redundant components are deliberately varied to allow individual components to emphasize desired characteristics, with the intent of forming a complementary set. Components can be crafted to optimize performance, even at the risk of errors, if a backup component can be relied upon to provide enough functionality to keep the system safe. Similarly, the backup component can focus on reliability and simplicity since it functions only infrequently, mostly in case of failure. Analytic redundancy is particularly applicable to systems where software must be upgraded. Use of a well-proven backup component allows new functionality to be safely introduced and tested during system operation.

During operation, Simplex ensures the integrity of the overall system by using the Simple Leadership Protocol [Sha 95]. Of all the analytically redundant components available, only the *leader* is in actual control. The remaining components are held as spares. The leader remains in control until it either fails an error check or is replaced by the system operator.

A Simplex-based system is fault resistant because the system includes a set of checks on the behavior of the constituent elements and switching logic to select among the analytically redundant components when the checks detect a leader’s failure. The specific checks applied are system dependent.

Analytic redundancy is applied to the Coordinated Prototype to ensure the following properties:

- **The pendulum remains upright.** The pendulum will become uncontrollable (potentially) if it gets too close to the end of its track, if it is allowed to tilt too far, or for other reasons. By running a trusted controller in an analytically redundant manner we can guarantee that these conditions will never be allowed to occur.

- **The pendulum motion is coordinated.** The two pendulms are kept within a specified distance of each other. In the demonstration, a lightweight rod can be laid across the tops of the two pendulums and will stay balanced during communications attacks.
- **The pendulum controllers can be upgraded.** Pendulum controllers can be changed during system operation, with any errors in the new software being contained safely.

3.2.1 Analytic Redundancy as Applied to Control

In a typical control system, it is impossible to have good coverage tests of the correctness of instantaneous outputs and it is often not possible to roll back the system even if the output is later recognized as incorrect. For example, in an inverted pendulum, if the pendulum starts to fall, at some point it will be unrecoverable no matter what action we take. Thus, we require that while the system is under the control of the high performance software it always be recoverable by the simpler high assurance software. This means that the high performance and the high assurance software are not logically independent. Rather, the high performance upgrade and baseline controllers must keep the system states within the recoverable region of the simpler high assurance safety controller. A comparison between the difference in reliability and availability of three different types of analytically redundant control systems, including Simplex, alternatives are given in Appendix B.

Once we are assured that the system is recoverable by the high-assurance safety controller, we can add optimization steps such as performance monitoring. In the context of motion control, performance monitoring can be achieved by monitoring the square of the errors between the reference trajectory and the system trajectory over a moving window of time. Simple output range checking is also used to speed up the error detection, but we do not rely upon it since its fault coverage is limited.

We execute different controllers in parallel, but we prefer the outputs from the high-performance upgrade controller as long as the system performance is acceptable and the system's state is within the recovery region of the high-assurance safety controller. If either the output range checking or the performance monitoring indicates there is an error but the system state is far from the boundary of the recovery region, the control can be directly passed to an alternate (baseline) high performance controller. Otherwise, the high assurance safety controller is invoked to move the system sufficiently far away from the boundary at which point control is passed to the baseline high performance controller.

3.2.2 The Decision Module

The behavior checking and switching logic is normally placed in a *decision module*, which typically includes the testing code and the controller of last resort (the high assurance safety controller). Since both elements must be functional for safe operation, their inclusion in a single process simplifies system scheduling without decreasing system safety.

For the Coordinated Prototype, two separate decision modules are present on each of the prototype's maneuver systems, the inner decision module to monitor the inner loop controllers—responsible for direct pendulum control and the outer decision module to monitor the outer loop coordinators—responsible for coordination of the pendulums. In the Coordinated Prototype, each decision module (both inner and outer) contains two controllers, a high assurance or safety controller and a baseline high performance controller. Operationally, the prototype limited dynamic upgrades to a separate computer system, and therefore there was little reason to place trusted controllers and coordinators into separate replacement units.

Two factors were important in devising checks for the prototype decision modules:

1. Is the controller or coordinator exhibiting incorrect behavior, by consuming excessive resources, failing to respond in a timely fashion or producing incorrect output?
2. If the output from an controller or coordinator is used to control the system, will the system enter a state where the remaining analytically redundant elements cannot successfully control the system?

While the problems of developing and implementing appropriate switching logic are considerable, the problem can be simplified by considering a sufficient solution rather than an optimal solution. In the prototype, it is critical that the "safety controller" be able to recover the system after a failure in the lead controller. It is difficult to determine the full recovery range of the safety controller, but a partial range is sufficient provided that

- The partial range is not incorrectly selected.
- The system meets operational goals (is fast enough, accurate enough, etc.). The prototype used safety regions based on empirical testing and controller modeling to determine a sufficient recovery zone for the safety controller and coordinator.

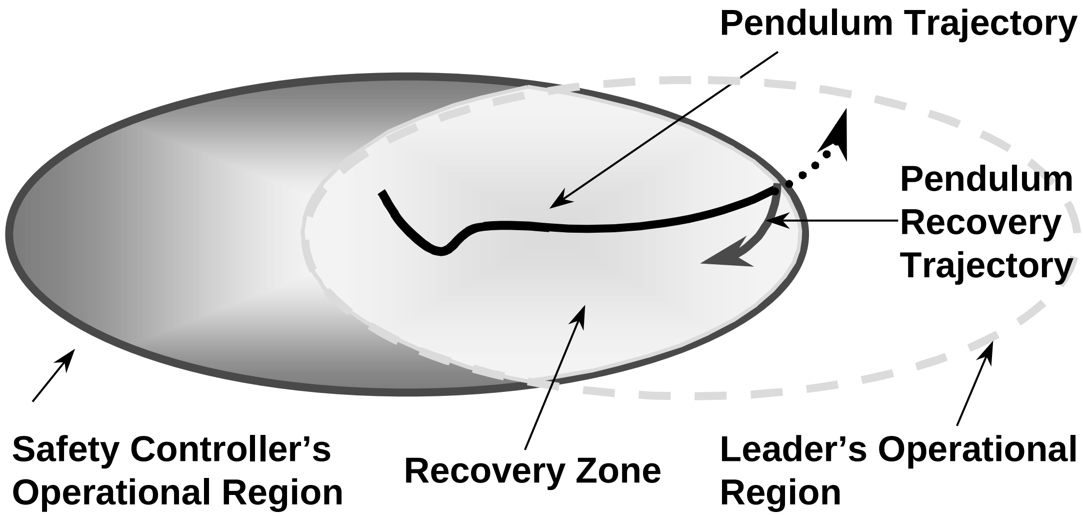

Pendulum Trajectory
Pendulum Recovery Trajectory
Safety Controller's Operational Region
Recovery Zone
Leader's Operational Region

Figure 2: Recovery Zone for Two Analytically Redundant Controllers

3.2.3 Trusted and Untrusted Elements

Intrinsic, in the use of analytical redundancy, is the concept of placing higher confidence in some elements than others. This confidence is not absolute, but when two element's results disagree, the final arbitration is based on the degree of trust placed in the contending elements.

Replacement units are trusted because of long experience using them, extensive testing, or formal proof of correctness. Untrusted units are generally new or upgraded versions of older trusted units. Trusted units replace untrusted units upon the occurrence of an error in the untrusted units.

In designing the prototype, the controllers and coordinators containing analytically redundant controllers were split into *trusted* or *untrusted* groups. This distinction gives an indication of confidence in the analytical sense, but it is primarily used to distinguish between those controllers whose origins are well known and controllers which are new, perhaps transmitted through insecure links and in which much hope but little trust is placed.

In the prototype, untrusted replacement units are placed on a separate node precisely because they are more likely to be faulty, and are potentially a source of deliberately designed attack. COTS-based nodes are considered vulnerable to a deliberate and considered attack, but node failure is not capable of causing system failure due to the firewall provided at the communications link.

3.2.4 Software Upgrade

For conventional software upgrades, new code is written, tested “sufficiently”, and then introduced into a running system. This process depends heavily on the accuracy and completeness of the testing process before software is used in actual operation. In the Simplex Engineering Framework, the conventional software upgrade process is augmented by provision for the safe introduction of software upgrades into an operational system.

Replacement units allow the safe upgrade of the system during execution without precluding the use of conventional upgrade techniques. Since the real-time control and coordination tasks were key elements of the prototype, they were selected for online upgrade and written as replacement units.

Up to three analytically redundant versions were provided for real-time control of the inverted pendulum and three for the pendulum coordination software. Of these, the upgrade controller is encapsulated in a replacement unit and isolated on a separate node and is upgradeable during system operation. The last chance safety controller and the medium performance baseline controller were, for the coordinated prototype, “hard-wired” into the decision module. The decision module can only be upgraded in a conventional manner.

3.2.5 Analytic Redundancy Compared to Other Methods

Analytic redundancy is related to both recovery blocks and N-Version programming [Randell 75 and Avizienes 85]. To summarize, in a recovery block system, a program unit is executed and then an acceptance test is applied. In the event that the acceptance test fails, the system is rolled back to a recovery point and an alternate program unit is executed. The sequence (execute—apply acceptance test—rollback—execute alternative) is repeated until either the acceptance test is passed or there are no additional alternates. This is an example of backward recovery. In N-Version programming, multiple presumably independent program units are developed to accomplish the same task via (perhaps) differing algorithms. The multiple units are executed in parallel and a majority determines the correct set of outputs. This is an example of fault masking. In analytic redundancy, we execute multiple presumably independent program units in parallel, but we prefer the outputs from one of the program units as long as the output passes the acceptance test. If the acceptance test fails, we use the output of one of the alternative program units. Because it does not require rollback, analytic redundancy is an example of a forward recovery method.

3.3 Rate-Monotonic Scheduling Theory

For real-time systems, we also have to be able to make performance guarantees. Generalized Rate-Monotonic Scheduling (GRMS) Theory [Sha 94] guarantees that, as long as certain conditions are met (e.g., task execution time, strict adherence to priority, etc.), a given task set can be guaranteed to meet its deadlines. Rate Monotonic Analysis (RMA) techniques coupled with the appropriate real-time operating system scheduling support can provide both analytic and runtime guarantees for real-time applications. Simplex utilizes RMA and GRMS techniques to ensure predictable execution, including upgrades of software components during system operation. A full description of RMS is beyond the scope of this paper. For a general introduction, see the article by Sha and Goodenough [Sha 90]. A practitioner's guide is available in the book by Klein [Klein 93].

RMS Theory was applied to the Coordinated Prototype to ensure that the pendulums continue to meet all real-time constraints in their operation even in the presence of faults. Under RMS, higher frequency tasks are assigned higher priorities. Within the same frequency, ties can be broken arbitrarily without affecting schedulability.

A simple example may help in understanding RMS. In this example, there are two task frequencies. The pendulum controller is executed at 50 Hz while the coordinated outer loop is running at 10 Hz. The 50 Hz tasks are given higher priorities than the 10 Hz tasks according RMS. Note that RMS requires priority inheritance. Any POSIX .1b OS such as Lynx OS automatically enforces priority inheritance.

Within the 50 Hz controller, the high assurance controller and the decision logic are given the relatively higher priority than the untrusted high performance control software. This simple arrangement ensures that the high assurance controller and the decision logic can meet their deadlines even if the high performance controller overuses its budget.

Note that this arrangement is adequate for the case where there is only a single control loop. Should there be multiple, hard, real-time controllers in the system, the execution time of the controllers must be monitored during runtime and the lower priority tasks must be interrupted if they overrun their execution budget. If not, the high performance controller could miss its deadline, and if lower priority high real-time controller(s) are present, their timeliness could be adversely affected. As long as the replacement unit can be terminated when it reaches its execution limit, the system remains schedulable and will meet its deadlines.

3.4 Simplex and COTS

Enabling technologies critical to the Simplex engineering framework can be crafted as part of the system, but the cost of constructing a complete Simplex-system on a “bare” computer would be very high. Throughout the evolution of Simplex based systems, support provided by commercially operating systems and use of standard hardware for Simplex has been essential.

For example, Simplex Engineering Framework depends on certain system protections to isolate replacement units and allow fault recovery. These protections include the following:

- **Address space protection**–to prevent processes from affecting memory dedicated to other processes
- **Execution time monitoring–**to detect when processes are exceeding their execution budget
- **Process creation and deletion**–to allow replacement units to be created and destroyed

These services are available in some COTS operating systems and hardware. The Coordinated Prototype used COTS support for these features. Simplex is compatible with the real-time POSIX standard and with operating systems that implement the RT POSIX standard.

3.5 The Benefits of Simplex

The Coordinated Prototype uses the Simplex Architectural Framework to provide two major benefits:

1. Failure resistance
   - Combine high reliability with COTS
   - New features without loss of reliability
2. Software upgrade capability
   - Separation of concerns
   - Reduce test and validation requirements

The prototype requires that the system handle software failures without halting operation. Simplex provides a model for handling faults that can be applied to a complete system or to targeted portions of a system. In the prototype, selected portions of the system are made resistant to software faults. Since Simplex is compatible with the use of COTS software and hardware, the prototype uses COTS components for all the prototype hardware and most of the prototype software.

Simplex provides techniques to support software and hardware upgrades by allowing upgrades of the system to take place without compromising system safety and reliability. In the prototype, deliberate errors introduced into upgraded pendulum-control algorithms are handled without failure. By separating out a safety critical core of the system from complex, high efficiency system functions, upgrade costs can be reduced particularly when testing can be safely accomplished in the operational system.

4 Prototype Design and Implementation

The Coordinated Prototype was designed and implemented as a concept demonstrator using the Simplex engineering framework.

4.1 Technical Challenges

The prototype was required to meet several technical criteria:

- It must be highly reliable. Specifically, it must provide continuous control for the real-time portions of the system; interruptions in communication, software flaws and upgrade failures must be handled without interruption of real-time control.
- The effect of communications attacks must be contained without failure and a recovery strategy must be employed when communciations are resumed. Attacks made by bombarding the target system with messages and attacks that prevent the delivery of legitimate messages must both be handled.
- It must allow software to be changed and upgraded before and during operation. Faults in the upgraded software must be handled without preventing the continuous operation of the system, whether the faulty software halts, produces incorrect output, or runs wild.
- It should use COTS hardware and software wherever possible.
- Software from prior Simplex prototypes should be reused in whole and as design templates wherever appropriate.

Several simplifying assumptions were made to allow the prototype to meet the above criteria:

- While the pendulums must remain upright and within the specified distance of each other, coordinated movement may be slowed or stopped temporarily.
- Communications attacks may delay or change the ordering of messages and spurious messages may be generated, but the contents of legitimate messages may not be changed.
- Communications may only be blocked for a bounded (finite) length of time.
- Only selected software components may be upgraded; specifically, the algorithms for real-time control and coordinated motion may be changed, but communications and operating software need not be fault resistant.[1]

[1] Fully fault tolerant Simplex architected prototypes have been developed and demonstrated which allow dynamic upgrade of all software elements. This functionality was feasible for the prototype, but was not included to reduce costs and speed deployment.

4.2 Hardware and Software Architecture

The prototype consists of a set of personal computers (PCs) connected by communications links to real-time devices (the inverted pendulums) and to each other (Figure 3).

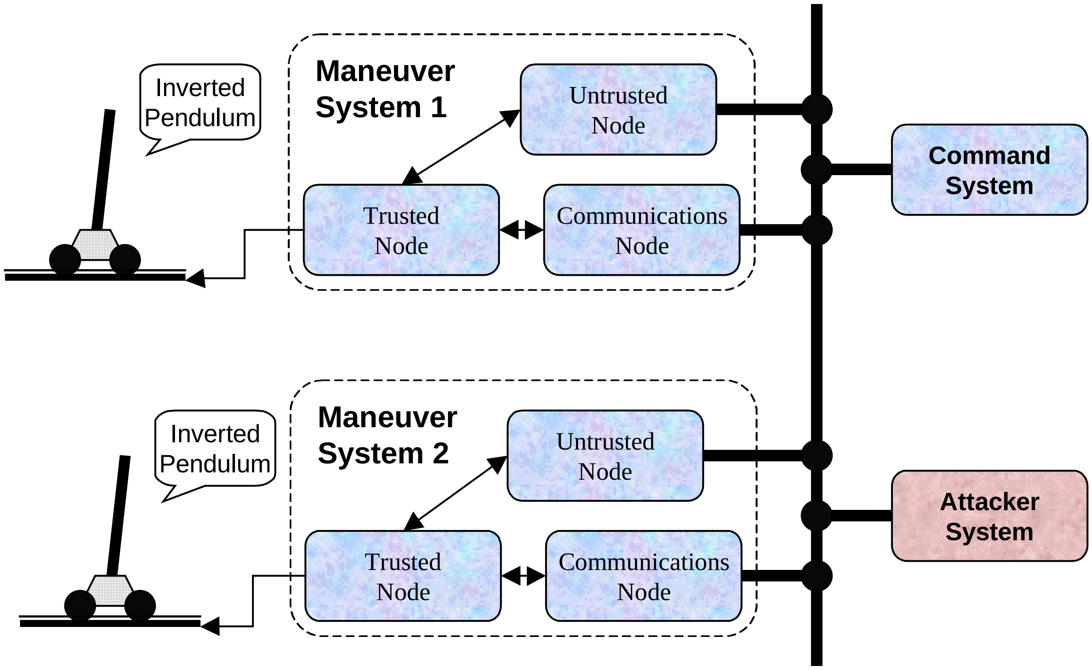


Figure 3: Hardware Configuration for the Prototype

The computers are grouped into four task specific systems. Each system represents a conceptually independent entity, which can reach other systems through external communications links:

- two maneuver systems, each with an attached real-time control device (notational “strike elements”)
- a command system that contains a user interface for operational control of the system and provides destination locations to the maneuver systems (the “command center”)
- a hostile jamming system that performs communication attacks (the “attacker”)[2]

While the command and jamming systems consist of a single computer, the maneuver systems have three computers (nodes) and an inverted pendulum. Each of the nodes in a maneuver system supports a specific function: running trusted software, handling external communications, and running untrusted software.

[2] Additional communications attack software is placed in the manuver systems to implement selective loss of legitimate messages (e.g., “50% of messages are lost”).

4.2.1 Real-time Device Control and Coordination

The Coordinated Prototype includes two (identical) inverted pendulums (Figure 4). Each pendulum consists of a track with a cart-mounted pendulum. The pendulum is a solid metal rod joined to the cart by a hinge that allows the rod to swing freely. The cart can be moved back and forth on the track by a motor attached to the cart. These devices are inherently unstable and need constant control input to keep the pendulum upright.[3] The prototype uses a modified commercially available pendulum with extended length tracks to allow greater cart movement.

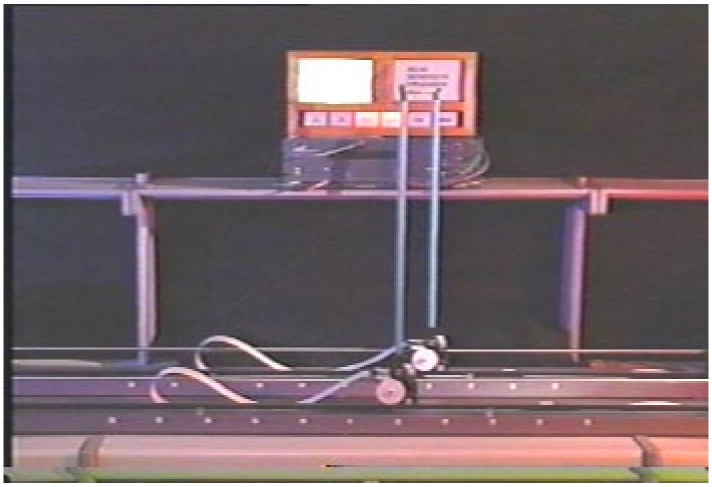


Figure 4: Inverted Pendulums

Initially, the rod is placed in a near vertical position by hand and the control program is started; the computer can then control the angle of the rod by moving the cart. The inverted pendulum will fall if the rod leans too far, if the cart reaches the track end, or if the rod angular velocity is too large. In addition to keeping the inverted pendulum upright the prototype must move the cart to specified and arbitrary locations along the track.

The unstable inverted pendulum is used in the prototype to represent, in a simple way, a system where a complete loss of the controlling computers will have a catastrophic effect. In the prototype, the system can lose some functionality due to attacks or faults but not at the risk of allowing the inverted pendulum to fall.

The ability to maintain safe real-time control of a single device in the presence of software and hardware faults and failures has been demonstrated in other Simplex-based prototypes; this prototype adds the capability of coordinating two real-time devices and doing so under communications attack. Coordinated motion is achieved through software and inter-system communication; the standard pendulum hardware was not modified. The two pendulums

[3] With sufficient care, inverted pendulums can be “parked” in a vertical position, but they seldom remain balanced for long due to their susceptabilty to external disturbances.

were placed side by side and, in coordinated mode, a lightweight horizontal rod placed across the rod tops will remain in place even when software faults and communications interruptions occur.

4.2.2 Computation and Communication Topology

The nodes in the prototype are tied by communications links. Each system shares a common external network, while communications between computers within the multi-node maneuver systems are handled by dedicated links. Conceptually, external communications are through a broadcast media subject to interception and to attacks that can block and delay message transmission. Internal communication links, by contrast, are private and safe from interference. In each maneuver system, one node is directly cabled to an inverted pendulum and is responsible for sending control output to the pendulum and reading pendulum position data (Figure 1 on page 8 and Figure 5 on page 26).

4.2.3 Software Components

Figure 1 shows a high-level view of the software. The network is shown as a heavy vertical line with connections to individual systems drawn horizontally. To the right are the attacker and command systems, which have a support function in the prototype. They use conventional software architectures. The maneuver systems, which have real-time and fault resistance requirements, applies the Simplex engineering framework to the software design.

The command system has the least complex software of the systems. The command system is responsible for interacting with the user, displaying the prototype status and issues commands to the remaining systems via the network. The command system is responsible for sending periodic commands, but is not responsible for real-time control.

The attacker system simulates wireless jamming by executing two types of denial of service communications attacks over the local network: message bombardment and message blocking. The attacker performs message bombardments by sending a barrage of packets over the network to consume available bandwidth and processing resources on receiving nodes. Message blocking is performed indirectly through “agent software” installed on the maneuver systems. In message blocking attacks, individual messages are either delivered or blocked according to the probability of receipt orders set by the system operator (through the command system or on the attacker). The attacker software is not fault resistant.

The maneuver systems are multi-node systems, with each node assigned a specific function. The three nodes are not required for computational power; instead the functional separation ensures system safety:

- The trusted node handles core real-time functions.
- The untrusted node provides a proving ground for software upgrades which may fail during operation.

- The communications node handles the external communications for the trusted node, isolating it from the network.

The software on the trusted node provides the critical core functions of the system: direct control of the inverted pendulum, core real-time control software, and core coordination between the maneuver systems. The trusted node is in direct communication with the other nodes of the maneuver system via point-to-point serial lines, but uses the communications node for external transmissions to the network.

The untrusted node supports upgraded real-time control and coordination software. Any fault in the upgrade, whether unintentional or malicious, is, at the worst, able to affect only the untrusted node. This design also allows the use of commercial operating systems, which might fail under a deliberate attack.

The communications node handles the external communications of the maneuver system. By isolating communications functions, communications attacks cannot overrun the critical functions of the maneuver systems by consuming excessive computing resources.

4.2.4 Communications Attack Modes

The Coordinated Prototype distinguishes between inter-system and intra-system communications. Inter-system communication is subject to interruption and interference through communication attacks, simulating wireless communications. Intra-system communications is immune from direct external interference.

Two basic types communications attacks were considered during the design of the prototype:

1. Denial of service attacks: legitimate messages are lost or delayed during transmission.
2. Deceptive jamming attacks: contents of messages are altered or false messages with correct syntax are created.

After review of possible attack modes, several limitations were placed on the allowable attacks to simplify design and implementation of the system:

- The contents of legitimate messages may not be changed.
- Communications may only be blocked for a bounded (finite) length of time.
- Only the maneuver systems will be attacked directly (the command system is affected indirectly by reduced network capacity).

These restrictions precluded deceptive jamming attacks, but allowed a guarantee of communications recovery during coordinated pendulum movement. Limiting attacks to the maneuver systems simplified the design and implementation of both the attacker and command systems.

The prototype implements two kinds of denial of service attacks:

1. Message blocking: legitimate messages are lost during transmission.
2. Message bombardment (or “packet barrage”) attacks: spurious messages are generated and directed at target communications links.

During message blocking attacks, each message may reach its destination or it may be lost during transmission according to probabilities set in a jamming specifier sent to agent software on the maneuver systems.

During a message bombardment, the attacker broadcasts network packets to the other systems in an attempt to clog the network. While these messages consume resources on the targeted systems, they are not deceptive or false messages.

Additional information on the communications attack implementation is available in Appendix A.

4.3 Prototype Implementation

In implementing the prototype, the development team exploited the software and experience from previous Simplex demonstrations and prototypes. Important components reused from previous Simplex systems include the commercial operating systems for each of the nodes, software development tools, code for pendulum control, and intra- and inter-processor communications software.

Major implementation challenges for the prototype included the following:

- **Long track pendulums**–these pendulums were unique to the prototype. The prototype real-time pendulum control software was based on the standard length pendulum controllers, but the long track pendulum dynamics required an extensive rewrite of existing software.
- **Pendulum coordination**–the requirement to coordinate the movement of two separate pendulums required an additional layer of coordination sofware running in real-time.
- **Communications support**–unlike previous Simplex prototpes, the prototype allowed disruption of communications links. This required creation of software to disrupt communications as well as the implementation of a separate set of secure communications links between nodes within systems.
- **User interface**–in the prototype, the user interface regulates coordinated motion by sending target locations to each maneuver system. This had the effect of making use of the user interface a requirement for coordinated motion. In previous Simplex systems, the user interface was simply a convenience for system operation.

During the implementation phase of the prototype, a small team of developers was used. The problem was largely separable individual tasks according to the expertise of the participants. An iterative design, implement, assess and redesign strategy was employed to implement the system. Relatively little specialized development software was used, although a memory logging facility developed for previous prototypes was found to be useful.

4.3.1 Long Track Pendulums

The long track pendulums were scaled up from standard pendulum hardware. Much of the hardware was adapted directly from standard pendulums. In order to accommodate the longer track length without use of new sensors, the granularity of track measurement was increased. Consequently, the accuracy of track position measurements for the pendulum cart was reduced.

Similarly, the extended pendulum length required that the drive system be extended by splicing together pieces of rack to engage the cart drive gear and the provision of longer cables to allow greater cart movement. Joints in the rack and the extra cable length were found to interfere with the pendulum movement. The pendulum hardware was modified by replacement of cables and switching from a gear drive system to a friction drive system.

The pendulum power supply, which provides regulated DC power to move the pendulum, was found to pass unwanted signals. A low pass filter was added to reduce interference. Variance in individual power supplies is also a factor in setting the pendulum controller gains for the individual maneuver systems.

In order to model pendulum behavior and create pendulum control software, a new set of parameters was required. These were determined empirically and the existing pendulum controllers were rewritten to work with the new pendulum hardware.

4.3.2 Pendulum Coordination

Pendulum coordination, the movement of two real-time devices in synchronization, was a new requirement for the prototype. The design of the coordination software and the algorithms employed in its preparation was a major element of the development. The software itself was based on the existing pendulum control software and presented relatively few implementation challenges, although the code was frequently modified during the course of software development.

4.3.3 Communications Support

The Coordinated Prototype consists of individual computers connected by communications links; within nodes, the modules of the Simplex architected system must intercommunicate to perform their assigned functions. This includes modules that are dynamically replaceable, which must be able to connect and disconnect from communications links during system operation.

The prototype design required the development of two types of system-to-system communications links, distinguished by their vulnerability to attack. Communications links between individual systems were subject to interference. Communications within systems were immune from interference.

In single node systems, internal communications links could be handled with inter-process communications based on system services and preexisting Dtag software. The maneuver systems, with multiple nodes, required secure links between the system nodes in addition to inter-process communications.

4.3.3.1 Dtag Communications

A communication abstraction, Dtag communications is used to link the dynamically replaceable components. The Dtag interface allows a common interface for replacement units, whether they are located locally (on the trusted node) or on a separate node (the upgrade node).

The Coordinated Prototype makes use of Dtag communications to support dynamic component binding. Dtag communications is a form of publisher-subscriber messaging analogous to POSIX .21 labeled messaging. In Dtag communications, communications ports are labeled (or named) with **tags**. To establish communications, a subscription request for a specific tag is made by the subscriber to a **tag manager**. The tag manager establishes the connection to the publisher who holds the specified tag. The requestor does not need to know where the tagged communications port actually resides and the responding publisher does not know how many subscribers are using a tag.

The tag manager maintains the list of tags and the locations of communicating processes. Limits can be set on how many subscribers are allow to subscribe to a specified tag.

For efficiency, the prototype uses Dtag communications to handle connections not to distribute individual messages. To communicate through a specific tag, a connection must be established which gives access to the stream of messages passing to or from the tagged port. There is no provision to send individual messages to a list of tags.

4.3.3.2 Inter-node Communications

Two different media were implemented for communications between nodes.

4.3.3.2.1 Between Systems

Ethernet links were used for inter-system communications, simulating the characteristics of a radio frequency broadcast media. Standard Ethernet software was used for basic communications. Additional software was added to some message queues to selectively block receipt and transmission of messages. A separate computer system was used to handle message bombardment (packet barrage) attacks.

One node of the maneuver system, the trusted node, is not connected to the Ethernet, protecting it from the direct effects of network attacks. It can communicate indirectly through the communications node.

4.3.3.2.2 Within Maneuver Systems

With network links subject to attack, separate communications links were implemented between nodes that made up the maneuver systems. Standard serial hardware was used for this purpose. Software for the serial links was developed to support Dtag communications through serial connections. Due to the limited capacity of the serial links, some messages between maneuver system nodes are transmitted between the untrusted and communications nodes via the Ethernet (Figure 5).

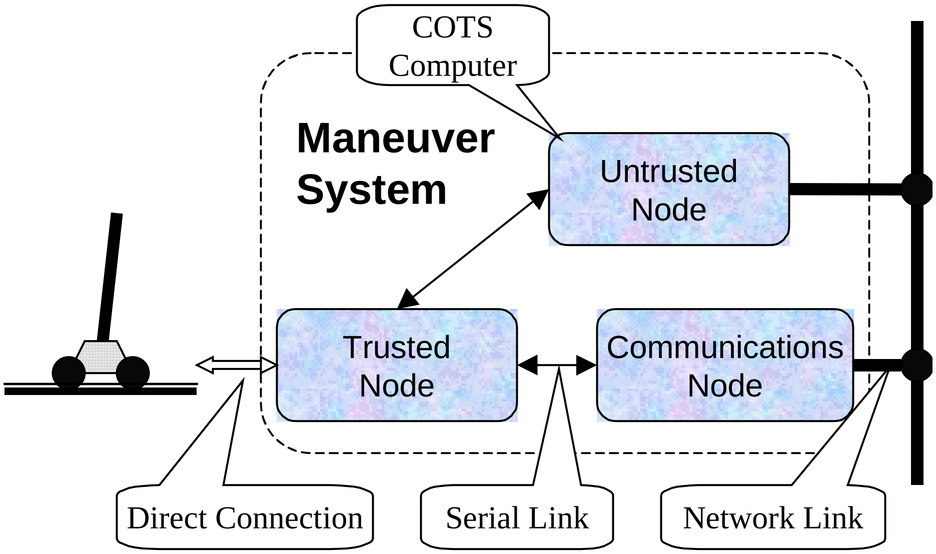


Figure 5: Communications Links for a Maneuver System

4.3.3.3 Intra-node Communications

The Coordinated Prototype uses the interprocess communications provided by the operating system. The communications blocking software and Dtag communications facilities are built on top of the operating system facilities.

4.3.3.4 Replacement Communications

In the Coordinated Prototype, Dtag communications are the basis for allowing dynamic alteration of software. Using Dtag as a foundation, replacement communications implements replacement unit creation and destruction.

In the prototype, replacement units encapsulate the pendulum upgrade controller and upgrade coordinator, which are located on the untrusted node of the maneuver system.

The **replacement unit manager** carries out the actual activation of replacement units. It handles process creation, synchronization, and retirement. During replacement unit activation, it must allow for simultaneous execution of functionally identical software, as the new software is readied and the old software is retired.

4.3.4 Control and Coordination Algorithms

The prototype divides the real-time control of the dual pendulum system into two parts: the control of individual pendulums and the coordination of the pendulums in synchronized motion. These two parts form a hierarchical structure, namely, an outer loop control which deals with the coordination and an inner loop control that keeps the pendulum standing up and moves it to a desired track position. In this section, the system level control specifications will be described first, the problem of controlling a single long track pendulum is considered second, and then the coordination of two pendulums is addressed. Finally, we examine how the system responds to faults and communications disruptions.

4.3.4.1 System Level Control Specification

The prototype is designed to demonstrate the reconfigurability of a multi-level control system to carry out the mission in both normal and fault disrupted conditions. The control objective of the overall system is to steer the pendulums to a target position in coordinated motion or independently. Such an objective is achieved by designing an inner loop control and an outer loop control for each pendulum. The inner control loop, here referred to as *the controller*, is responsible for stabilizing the pendulum in the upright position at a given set point on the track, while the outer control loop, called *the coordinator*, generates the set points for the inner loop control. Coordination is accomplished by adjusting the set points for each pendulum and successful coordinated operation is demonstrated by a bar attached the pendulum tips. The following specifications are used in design of the system.

User Commands:

- TARGET – Set the target track position.
- COOR_ON – Turn on the coordination. The pendulums will move coordinately.
- COOR_OFF – Turn off the coordination. The pendulums will move independently.

To formally describe the controlled operation, three overall system states (OSS) are defined as

OSS = {INIT, INDEPENDENT, COORDINATED}

With

INIT – the initialization state of the overall system;
INDEPENDENT – moving and stabilizing the pendulums to the target independently;
COORDINATED – moving and stabilizing the pendulums to the target in coordination;

the state transition diagram is given in Figure 6.

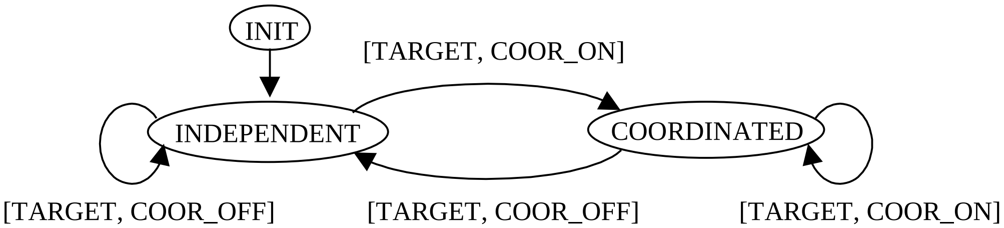


Figure 6: State Transition Diagram of Overall System Operation

Example: Suppose the pendulums are initially located at different positions along the track and they are required to move to a new target position in coordination. A sequential control procedure may be given by the user as follows: 1) Issue the commands TARGET and COOR_OFF to move the pendulums independently to a track position, say the mid-point of the two initial positions of the pendulums. 2) Once the pendulums are aligned, turn on the coordination with the command COOR_ON. 3) After the pendulums stabilized in synchronization, send the command TARGET to move the pendulums in coordination to the new target location.

Communication Assumption

Information on the current state of the pendulum and current status of the inner loop controller need to be exchanged between two trusted computers if COOR_ON is commanded.

Fault Model

- Communication fault: at least one trusted computer does not get a message from the other in the required time period.
- Pendulum fault: at least one of the pendulums is attacked by external disturbance and it is under the inner loop safety control.
- Alignment fault: the pendulums are too far from each other in the coordinated mode.

Safety Requirement

- Safety criterion for a pendulum: a pendulum is said to be *safe* if its state is inside the recovery region, otherwise it is *unsafe*.
- Safety criterion for coordination: coordination is said to be *safe* if
  - Both trusted computers receive messages at the specified rate;
  - None of the pendulums is under inner loop safety control;
  - The pendulums are aligned.

  Otherwise, the coordinator is said to be *unsafe*.

Performance Requirement

The pendulums carry the bar to the target position in a given time interval. Specifically, let MAX_SP_UPDATE and MIN_SP_UPDATE be the maximum and minimum set point distance updates in one outer loop sampling period, prev_sp and current_sp be the set points in the previous and current periods, respectively, and Target be the target track position. Then the coordinator that generates the current_sp is said to be *performing* if

current_sp = Target
if |prev_sp – Target| <= MAX_SP_UPDATE;
prev_sp + MIN_SP_UPDATE <= current_sp <= prev_sp + MAX_SP_UPDATE
if prev_sp < Target – MAX_SP_UPDATE;
prev_sp – MAX_SP_UPDATE <= current_sp <= prev_sp – MIN_SP_UPDATE
if prev_sp > Target + MAX_SP_UPDATE.

Otherwise, the coordinator is said to be *non_performing*.

4.3.4.2 Control of An Individual Pendulum

As shown in Figure 3, the physical system consists of a cart driven by a DC motor and a pendulum attached to the cart. The cart can move along a horizontal track and the pendulum is able to rotate freely in the range of $[-30^\circ, 30^\circ]$ with respect to vertical in the vertical plan parallel to the track. There is no direct control applied to the pendulum. Both the cart position *x* and the pendulum angle θ are measurable through two potentiometers. The dynamics of the system is described by the state of the system, which consists of the cart position *y*, the cart velocity $\dot{y}$, the pendulum angle θ, and the pendulum angular velocity $\dot{\theta}$. The physical system has state and control constraints. Specifically, the cart position is restricted in the range [-0.7, 0.7] meters due to the length of the track, the maximum speed of the cart is 1.0 meter/second, the angle is constrained to the range of $[-30^\circ, 30^\circ]$, and the motor input voltage is limited in the range [-4.96, 4.96] volts. The control objective of the inverted pendulum system is to move the cart to the given set point along the track with the pendulum standing at the upright position (i.e., $\theta \approx 0$).

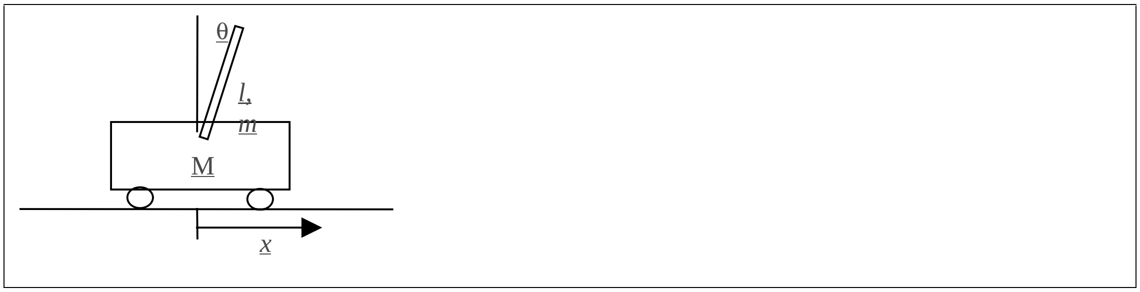


Figure 7: Single Pendulum Physical System

The dynamics of this system can be modeled by a 4-th order linear-differential equation in the form: $\dot{x} = Ax + Bu$, where $x = [x_1, x_2, x_3, x_4]^T$ is a vector of states consisting of the difference between the cart position and set point, cart velocity, pendulum angle and pendulum angular velocity and *u* is the control variable, the voltage sent to the motor. Matrices *A* and *B* contain the parameters of the physical system. The control algorithm is designed as a linear feedback control in the form $u = Kx$ with K a constant gain vector. Hence the control constraint can be converted to a state constraint and the set of state constraints considered in control design is given by

$$\left\{ \left|x_1\right| \le \min\left(|\,0.7 - sp\,|, |-0.7 - sp\,|\right)\mathrm{m},\ \left|x_2\right| \le 1.0\,\mathrm{m/sec},\ \left|x_3\right| \le 30^\circ,\ \left|Kx\right| \le 4.95\,\mathrm{volts} \right\}$$

with *sp* the given set point. In our design, we will move the cart in between [-0.5m, 0.5m] (i.e., have the range of the set point in [-0.5m, 0.5m]). Apparently, the constraint on cart position is a function of the set point. For instance, if the set point is at 0.5m (-0.5m), the cart constraint would be $|\,x_1\,| \le 0.2\,\mathrm{m}$, while it is $|\,x_1\,| \le 0.3\,\mathrm{m}$ when *sp* = 0.4m (-0.4m). Such a variable constraint will make it difficult to derive a recovery region. To address this problem, we re-examine the meaning of a recovery region. By definition, a recovery region is a region in the system state space in which the safety controller is able to stabilize the system without violating the system constraints. In control of an inverted pendulum, we could always design the safety controller to stabilize the system at the track position where the safety controller first takes over the control. In other words, the set point for the safety controller is always the cart position when the safety controller started to control the system. With this safety control objective, we could say that, when the safety controller is in control, the track constraint is always $|\,x_1\,| \le 0.2\,\mathrm{m}$ given that the set point is in the range of [-0.5m, 0.5m]. In summary, the state constraints used in design of safety controller and derivation of recovery region is given by

$$\left\{ \left|x_1\right| \le 0.2\mathrm{m},\ \left|x_2\right| \le 1.0\,\mathrm{m/sec},\ \left|x_3\right| \le 30^\circ,\ \left|Kx\right| \le 4.95\ \mathrm{volts} \right\}$$

With a linear state feedback control algorithm, the design of a safety controller amounts to determining the control gain *K*. Both control gain *K* and the maximized recovery region can be derived from the linear matrix inequality approach [Boyd ND]. Such approach will give us a quadratic function in the form $x^T Qx = 1$ with Q a positive definite matrix that defines a 4 dimensional ellipsoid. The recoverable region, $R = \{x : x^T Qx \le 1\}$, is the set of the states inside the ellipsoid. For the particular pendulum system in our lab, a satisfactory control gain was derived as $K = [7.6, 13.54, 42.85, 8.25]$.

Figure 8 shows the projections of cart position and velocity as well as pendulum angle and angular velocity of the four dimensional ellipsoid.

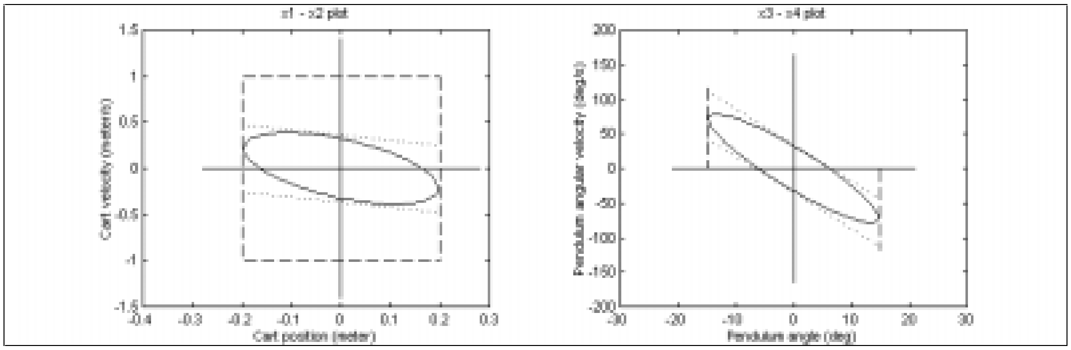


Figure 8: Recovery Region Projections

Solid lines–the boundary of the recovery region; Dashed lines–the state constraints; Dotted lines–the constraints due to control limitation.

The experimental demonstration system runs on a single PC using POSIX real-time OS, Lynx OS.

From the high-assurance computing perspective, what is relevant here is that each controller is a different software process. The high-performance control process-timing budget per sampling period is also monitored and controlled. When a process has a timing budget overrun fault, it will be detected and terminated, rather than denying the execution of other processes.

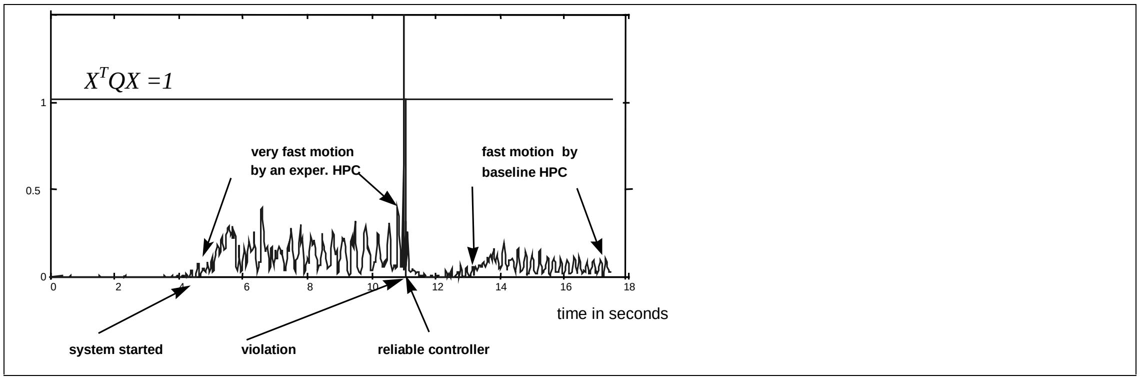


Figure 9: System Performance During Fault Recovery

A variety of faulty high-performance upgrade controllers have been tested. Predefined bugs include infinitive loops, sign errors, bad control gains, and designer bugs. A designer bug consists of malicious code that attempts to bring down the inverted pendulum with full knowledge of the high-assurance controller and the recovery region. They were designed by us for testing purposes and by visiting scientists to verify our claims. Upgrade controllers, with and without bugs, can be changed on the fly. In every instance, once the bugs were activated, the buggy upgrade controller would be switched off and the control passed back to the baseline controller once the high-assurance safety controller stabilizes the inverted pendulum.

Figure 9 is a sample of experimental data when a bug activates in a faulty upgrade controller. From time t = 4.3 to 11.2 second, we can see that x^TQx values form a relatively large wave pattern. This was due to the rod shaking back and forth modestly while the upgrade controller is moving it to the set point. But it is well within the recovery region. At time t = 11.2, the large spike of x^TQx value that is greater than 1 means that the system state would go outside of the ellipsoid $x^TQx = 1$, should we continue using the upgrade controller. This triggered the safety-switching rule that puts the safety controller back in control. The smaller x^TQx "waves" after the safety controller taking over means that the rod was shaking very little.

Finally, we want to mention the subject of performance monitoring. We should reject an upgrade controller if it performs poorly even if it stays within the recovery region. A controller's performance can be tracked during runtime, e.g., as a moving average of the squared of the error between the reference trajectory and the actual trajectory. If the moving average indicates that the upgrade controller performs worse than the existing one, we can switch control back to the previous baseline controller.

4.3.4.3 Motion Control of the Overall System

In the previous section, we have described how each pendulum can be controlled and recovered from faulty upgrade controllers. In this section we consider outer loop control, namely, control of the motion of both pendulums along the track. As described in the specifications, the pendulums can be commanded to move coordinately or independently; and in either case, the pendulum motion is controlled by generating a sequence of set points for the pendulum to follow. Therefore, the outer loop control is basically a set point generator for each moving objective of the pendulum in both normal and faulty situations.

To describe the system level behavior, we define four modes for the motion of the pendulums. By mode, we mean the type of operation that the system (subsystem) is undertaking. In our design, the overall system could be in the mode of independent motion, transitioning to coordinated motion, in coordinated motion, and transitioning to independent motion. Apparently, each mode may require different control strategy to carry out the operation. In the following discussion, we will concentrate on the coordinated motion mode and describe the outer loop control for both normal and fault tolerant operation.

In the experimental set up, each of the two pendulums' horizontal positions must be within ± 2 inches of the set point in order not to interfere with each other's operations. During normal operation, this is not a problem as long as each of them closely follows the two nearly identical reference trajectories given by a sequence of set points. However, there are two potential complications:

1. How can we ensure that both pendulums will receive the nearly identical destination information so that each can generate reference trajectories that are nearly identical, giving that the communication can be jammed?

2. When one of the upgrade controllers is faulty and enters a recovery mode, how can the other pendulum slow down (or stop) to wait for the other pendulum, given that communication can be jammed.

The first problem is overcome by a target-generation. Suppose that the current position of both pendulums is at the center and the operator wants them to go 8 inches to the right. Since adversary can jam one of the channels, it is possible that only one of them will receive the new destination command. To avoid the two pendulums having destination commands that are far apart, a destination command loop is used as follows. The operator station decomposes an operator's target command into multiple increments, for example, 16 steps of 0.5-inch increments. Initially, a destination of 0.5 inch to the right is sent to both pendulums. If acknowledgement from both pendulums is received, the operator station updates the destination command to 1.0 inch to the right and so on and so forth until the final destination command 8 inches to the right is sent. This way, no matter when communications are interrupted, the destination command received by both pendulums can differ at most 0.5 inches. It is important to note that the destination loop is independent of the physical movement of the pendulums.

Having ensured the error bounds on the target destinations received by both pendulums, we next consider the bound on the coordination errors during the movement. This is the primary issue that needs to be addressed in fault tolerance design of the outer loop coordination. There are two analytically redundant modes of coordinated movements. The baseline mode will use the trusted baseline controller together with a conservative (slower) reference trajectory generation algorithm. Under this baseline mode, each of the two pendulums must follow its local reference trajectory closely. Thus, coordinated movements can be ensured without explicit communication between the two pendulums. This baseline mode is a key to resist the communication attacks. As long as the target destination is sent to the two pendulums, coordinated motions can be ensured.

When high-performance upgrade controllers control both pendulums, the pendulums can carry out the coordinated movements in about twice the speed. However, this requires that both pendulums are using the upgrade controller and both upgrade controllers are working properly – that is both pendulums are using the upgrade controller and the two pendulums are in fact moving in close coordination. This is checked every 100 msec by message exchange. If both pendulums cannot confirm this condition, they switch into the baseline mode to carry out the last successfully received command.

The Simplex architecture with high-assurance operation mode and high-performance operation model described above crucially depends on the switching among the safety coordinator, the baseline coordinator, and the upgrade coordinator. In the following sections, we describe the detailed switching rules for the outer loop control in terms of the states of the coordinators and the states of the active coordinator, as well as the state transitions.

4.3.4.3.1 Coordinator States

Each coordinator has states defined as

CoorState = { ALIVE, TERMINATED, DISABLED }

Where

ALIVE – the coordinator is running and its output is enabled;

DISABLED – the coordinator is running but its output is disabled;

TERMINATED – the coordinator is terminated by the user.

A coordinator's state transition is determined by several factors, such as the user's commands, the selection of the active coordinator, coordinator's performance, coordinator's safety, and the timing performance of the coordinator. In our design, we make the following assumptions regarding the state of a coordinator:

1. The user can control the state of a coordinator by issuing one of the commands:

   Create / Kill / Enable / Disable the coordinator.

2. The coordination safety, coordinator's performance, and its timing performance only affect the state of the coordinator when it is active, namely

   - When the coordinator is the active coordinator, it will become disabled if the coordinator is unsafe, does not perform, or/and misses its deadline;
   - When the coordinator is not the active coordinator, the coordination safety, the coordinator's performance and its timing performance have no effect.

Based on these assumptions, the state transition diagram is given by

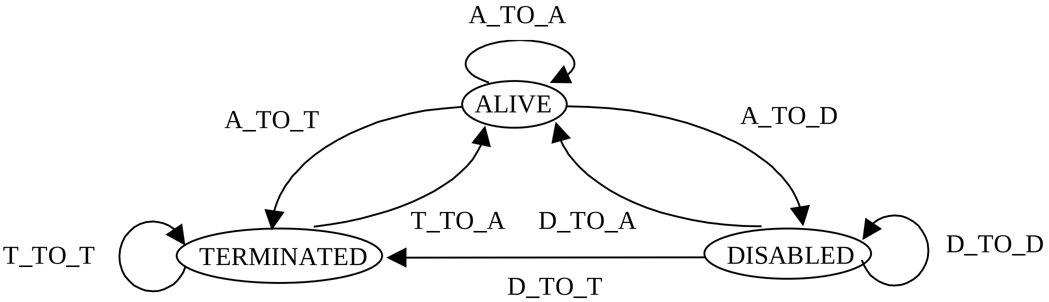


Figure 10: Coordinator State Transition Diagram

Where

- A_TO_D is true if the user disables the coordinator, or it is active and some of the following occur: coordination is unsafe, it does not performance, or it misses the deadline;
- A_TO_T is true if the user kills the coordinator;

- A_TO_A is true if both A_TO_D and A_TO_T are false;
- D_TO_A is true if the user enables the coordinator;
- D_TO_T is true if the user kills the coordinator;
- D_TO_D is true if both D_TO_A and D_TO_T are false;
- T_TO_A is true if the user creates the coordinator;
- T_TO_T is true if T_TO_A is false.

4.3.4.3.2 The States and State Transition of the Active Coordinator

The active coordinator has three states: Safety Coordinator, Baseline Coordinator, and Upgrade Coordinator. The state transition of the active coordinator is determined by the coordination safety, the states of the baseline coordinator and the upgrade coordinator. The detailed state transition diagram is given by:

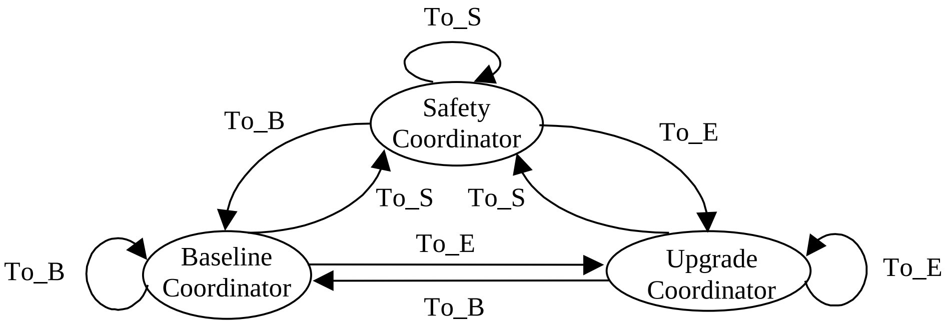


Figure 11: Active Controller State Transition Diagram

Where

- To_B is true if coordination is safe, the baseline coordinator is ALIVE, and the upgrade coordinator is either disabled or terminated;
- To_E is true if coordination is safe and the upgrade coordinator is ALIVE;
- To_S is true if coordination is unsafe, or coordination is safe but neither the baseline coordinator or the upgrade coordinator is ALIVE.

4.3.4.3.3 The Safety Coordinator

The safety coordinator has four states corresponding to four control modes. These states are defined as: No_Com, No_Pen, No_Align, and Transport, with

No_Com – dealing with a communication fault;

No-Pen – dealing with an unsafe pendulum;

No_Align – dealing with an alignment fault;

Transport – transporting when there is no fault and only the safety coordinator is available.

The state transition diagram for the safety coordinator is given by

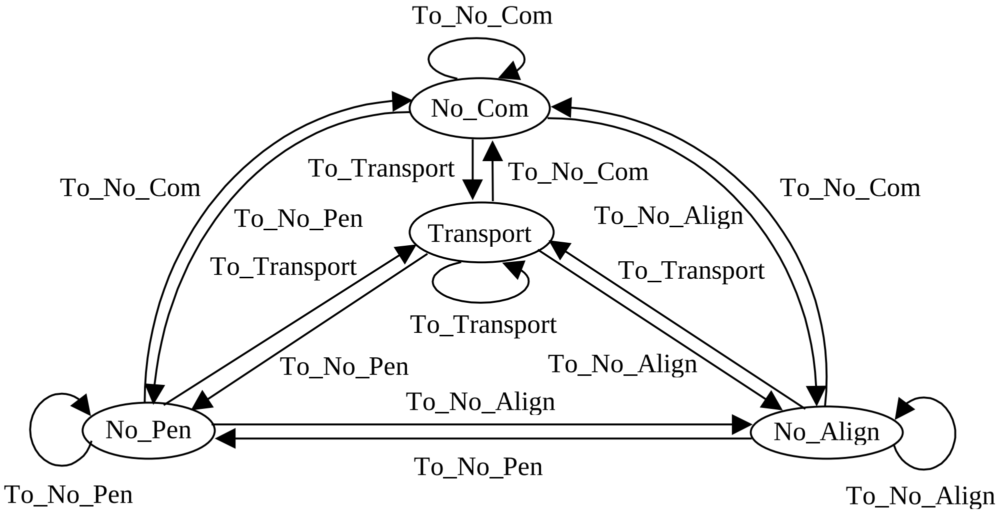


Figure 12: Safety Coordinator State Transition Diagram

Where

- To_No_Com is true if messages are blocked in at least one direction;
- To_No_Pen is true if communication is OK but at least one of the pendulums is under inner loop safety control;
- To_No_Align is true if both communication is OK and the pendulums are safe but the pendulums are out of alignment;
- To_Transport is true if communication is OK, the pendulums are safe and aligned.

4.3.4.3.4 A State Transition Diagram for the Overall System

We now integrate all the pieces presented above together to complete the design of the overall system. The following figure shows a state transition diagram for the overall system, which is used for implementation of the pendulum motion control.

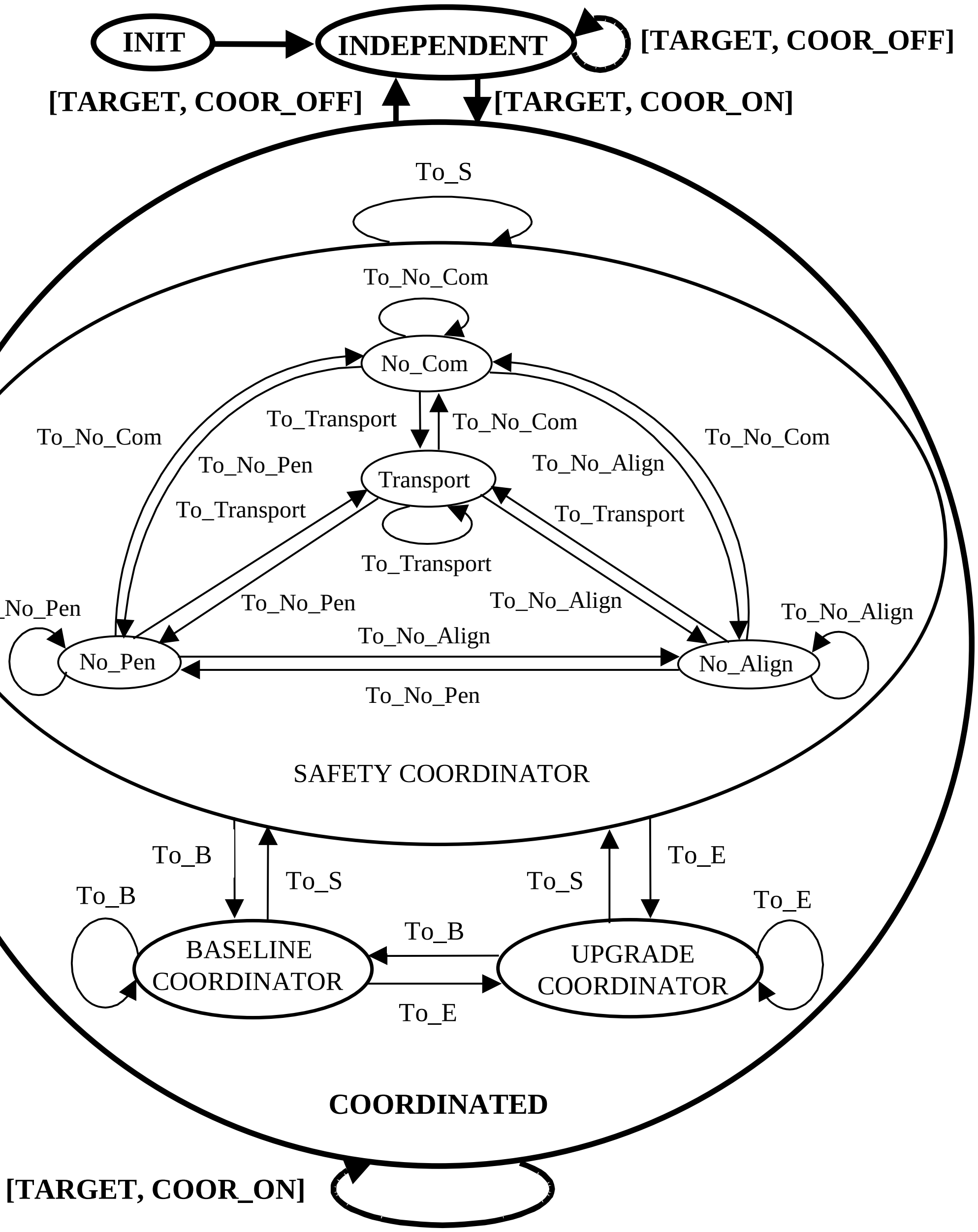


Figure 13: Overall State Transition Diagram

4.3.5 User Interface

The graphical user interface (GUI) provides a unified view of the Coordinated Prototype status and a convenient way to send commands to the individual systems from a single console. This functionality had been provided for other Simplex prototypes, proving a useful supplement to running individual nodes from local consoles.

The most significant services the user interface provides are commands to initiate (and stop) coordinated motion and to set movement goals for the maneuver system pendulums. When in coordinated mode, the interface regulates coordinated motion by sending target locations to each maneuver system in a series of small increments, waiting for acknowledgement from both maneuver systems prior to sending the next step. This protocol ensures that the pendulum coordinators do not become unsynchronized due to large differences in their individually maintained target locations.

4.4 Design Evolution

The Coordinated Prototype extended the application of Simplex technology to a new problem space of coordinated real-time movement in a hostile communications environment. During the creation of the prototype, the design changed and evolved, as performance requirements and implementation limitations were better understood. This section describes evolution of the design rational in response to changes driven by new requirements and the lessons learned during the implementation process.

Several prior Simplex prototypes used inverted pendulums and COTS hardware and software. These prototypes provided a base from which unique features of the Coordinated Prototype were designed and implemented. The elements from previous Simplex prototypes that were particularly applicable to the prototype design include the following:

- **Replacement units**–The replacement unit concept and its implementation were thoroughly explored in previous Simplex prototypes. The concept was successfully applied to the inner control loops with little modification. The replacement unit idiom was also applicable to the design and implementation of the outer (or coordination) loops used to synchronize pendulum movements.

- **Dtag communications**–The prototype used Dtag communications for intra-node and intra-system communications on the maneuver systems to handle connections between processes, for both replacement units and static processes. Extensions were made to support serial communications links and to provide jamming support. The Dtag implementation also required rework to support changes to the queue semantics in a new release of the COTS operating system used.

- **Inverted pendulum real-time control software**–Although the new pendulum hardware required new controllers, the existing software provided the basis for designing the new software.

- **GUI**–The existing GUIs provided a model for the design of the new interface.

The prototype design was iterative; a series of initial design meetings outlined the overall goals and requirements for the prototype and an overall design was prepared. Implementation of the prototype started with a focus on novel and essential features of the prototype. Implementation provided feedback on the feasibility of the design. The design was then modified to produce working software.

In parallel with the design modifications driven by implementation considerations, the overall design was also altered in response to reconsideration of the prototype objectives. The prototype is used as a demonstration vehicle and discussion with demonstration participants, particularly those with domain expertise served to refocus the prototype design.

The prototype embodies the final design, but incompletely documents the history of the design. In this section, focus is placed on the way the design evolved as well as the final form of the design.

4.4.1 System Topology

The Coordinated Prototype uses a set of intercommunicating computers. The relatively large number of individual computers was not a method of increasing computational power. At the initial design stage, it was realized that by dedicating individual computers to specific tasks the general programming of the prototype would be simplified because interaction effects would be reduced and the individual computers could be tailored to the task (e.g., by varying the operating system type between systems with stringent real-time requirements and those with a focus on user interaction).

Similarly, the requirement for using COTS software and hardware has the effect of introducing components whose resistance to failure is not completely testable. In the prototype, isolating the most vulnerable software on a separate system provided the final protection against software failure. The communications links provided a way of rigidly isolating the effects of errors to a single system.

The Coordinated Prototype did not attempt to provide hardware redundancy in the traditional sense; the focus was on containing and correcting software faults. (Hardware redundant systems had been previously demonstrated within the Simplex Engineering Framework.)

4.4.2 Communications

The communications requirements for the prototype design included several novel elements. In consequence, prototype communications topology and implementation underwent considerable change during the design and implementation of the prototype.

The communications design required are as follows:

- separate “jammable” and “unjammable” communications links
- real-time data transfer between nodes of the maneuver systems for pendulum controls.
- support for replacement units.
- communications attacks

4.4.2.1 Separate Communications Links

The need to allow communications attacks on systems while keeping the internal communications between a system’s nodes secure lead to the use of separate communications links for intra- and inter-system communication.

4.4.2.1.1 Intra-system Communication

With network links subject to attack, separate communications links were implemented for nodes that made up the maneuver systems. Standard serial hardware was used to provide point-to-point connections between the nodes of the maneuver systems. Software was developed to support proxy Dtag communications through serial connections.

Use of serial links was planned early in the implementation phase, if the throughput was sufficient. Implementation of the serial links proved challenging, since the serial link data rates proved insufficient to handle all the inter-node data across a single link. Simply reducing the data transfer rates did not solve the problem.

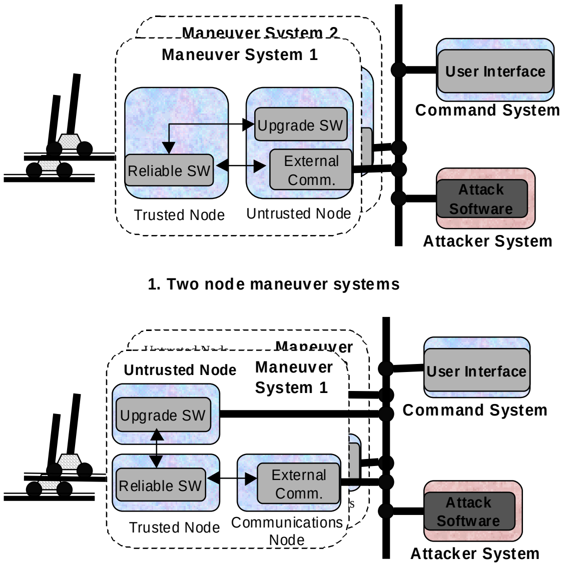
On the secure node hardware, a maximum of two serial lines could be supported when the system was configured with all required hardware. Using two serial connections, throughput was sufficient, but was less than the theoretical throughput for the equipment. This design, with adjustments to the data rates was sufficient for a two-node maneuver system. When a third node was added to the maneuver systems, some command and control data was rerouted to use network communications and the second serial link was used to connect to the new node.

4.4.2.1.2 Inter-system Communication

Inter-system communications required software that met the real-time goals of the maneuver systems, provided for communications attacks and allowed critical functions to be isolated from the communications attacks. An Ethernet network using the UDP/IP protocol was used for the inter-systems links. Ethernet hardware and software is not designed for guaranteed real-time performance, but prior experience showed that a lightly loaded network could be used successfully to close real-time control loops for the inverted pendulums used in the prototype.

Since the Ethernet software was COTS, its detailed structure was unknown and the prototype design assumed that communications attacks could place a heavy processing burden on any node connected to the network. This had the potential of interfering with other software on the system, which in turn might cause missed real-time deadlines.

The original maneuver system design called for a two-node design, with a trusted node handling core real-time functions and an upgrade node that would contain the network connection and also run upgraded software. The trusted node would use the upgrade node to provide network links in tunnel mode. However, the untrusted node is intended to run software that might be unreliable, perhaps even deliberately sabotaged. The possibility of the upgrade node receiving defective software and the consequent fault model led to the isolation of the trusted node's network connection onto a separate node.



1. Two node maneuver systems

2. Three node maneuver systems, with separate communications nodes

Figure 14: Evolution of the Maneuver System from Two to Three Nodes

4.4.2.2 Real-time Data Transfer

The Coordinated Prototype depends on the completion of several real-time loops for coordinated motion. Each maneuver system has an inner loop, which handles direct pendulum movement and an outer loop, which handles coordination between the two pendulums. Both of these loops span two nodes within the maneuver system. A coordination loop connects the two maneuver systems, allowing the outer (pendulum coordination) loops to exchange position updates and target commands. The user interface provides movement goals as a series of discrete steps to each of the maneuver systems; this update loop is timed, but the deadline is elastic in the sense that the pendulum can compensate by slowing down pendulum movement.

With the isolation of one of the replacement units for each type of controller on a separate node, communications throughput via the serial connection was a bottleneck to completing the two real-time maneuver system loops within one cycle. The first approach to this slowed the outer loop was to reduce communications volume. This did not increase throughput sufficiently and the controllers on the upgrade node were rewritten for a one-cycle lag (i.e., producing control output based on the system state projected forward in time).

During the development process, a distinction between the data exchanged in the real-time loops and commands from the user interface became apparent. When real-time data is lost or delayed beyond the current deadline, the correct recovery method is to use fresh data. With commands, sequencing is significant and it is important that the commands eventually be delivered in the correct order. This distinction was supported by adding an acknowledge/retry protocol to communications links from the user interface. Support at the Dtag level was considered, but not implemented in the present implementation.

4.4.2.3 Replacement Unit Support

Inter-component communications when active components are replaced must allow flexible splitting and redirection of data streams during real-time operation. The prototype uses Dtag communications to support replacement units. The preexisting Dtag software was enhanced to allow multiple nodes by adding support for serial links. Some investigation into alternate real-time networking protocols was performed (e.g. CORBA) but the existing Dtag routines were viewed as having a lower implementation cost for the prototype.

With the introduction of two real-time loops executing at different but harmonic rates, a variable rate delivery service was added to the Dtag routines. This allowed a subscriber to specify that it wished to receive 1 of n messages from a port. The primary use of this feature was to allow the outer loop to receive pendulum position data at a lower rate consistent with it’s slower execution rate.

4.4.2.4 Communications Attacks

The original communications attack design included two kinds of attacks: message barrages to consume bandwidth on the network and message blocking to prevent a selected percentage of messages from passing through a designated link (i.e., network connection to a single node). As the implementation proceeded, it was decided that blocking messages on a link did not afford sufficient discrimination and control. Consequently, the message blocking facility was recast to allow message blocking on individual communications sockets (e.g., rejecting messages on specific module to module connections).

4.4.3 Replacement Unit Creation

The term replacement unit is used to designate a “slot” where code may be replaced using dynamic component binding as well as the modules of code created for overlay into the slots. The following design and implementation steps have been useful in creating replacement unit modules

- Modularize the system into functional blocks of code.
- Identify blocks which should be dynamically replaceable. The ideal candidates have a single purpose, are likely to change frequently, and are “weakly” interconnected with the rest of the program (exchange relatively small amounts of data in a well understood manner).
- Define the data interface between the dynamically replaceable block and the rest of the system. The interface forms, in a sense, a contract between the replacement unit and the rest of the system with respect to input and output of information. It is useful to divide data items between required and optional items. Optional items provide for variation in the dynamic component, but need not be present for system operation.
- Identify the scheduling characteristics of the dynamic component. This requires determining the maximum resource a dynamic component can use before it is considered to be faulty.

The prototype design designated the upgrade pendulum controller and coordinator as replacement units early in the design process. Although preparation of the actual code proved challenging, the type and number of replacement units did not require revision.

To provide increased fault resistance while allowing the use of COTS software, one of the replacement units for the upgrade controller and coordinator was isolated on a separate node. With a serial communications link isolating the node, communications throughput was a bottleneck to completing the control loop within one cycle. The first approach to this problem used relaxed timing criteria for the outer coordination loop to reduce communications volume. This proved insufficient and the controllers on the upgrade node were rewritten for a one-cycle lag (i.e., to produce output based on the state of the system projected forward in time). This was sufficient to solve the communications bottleneck.

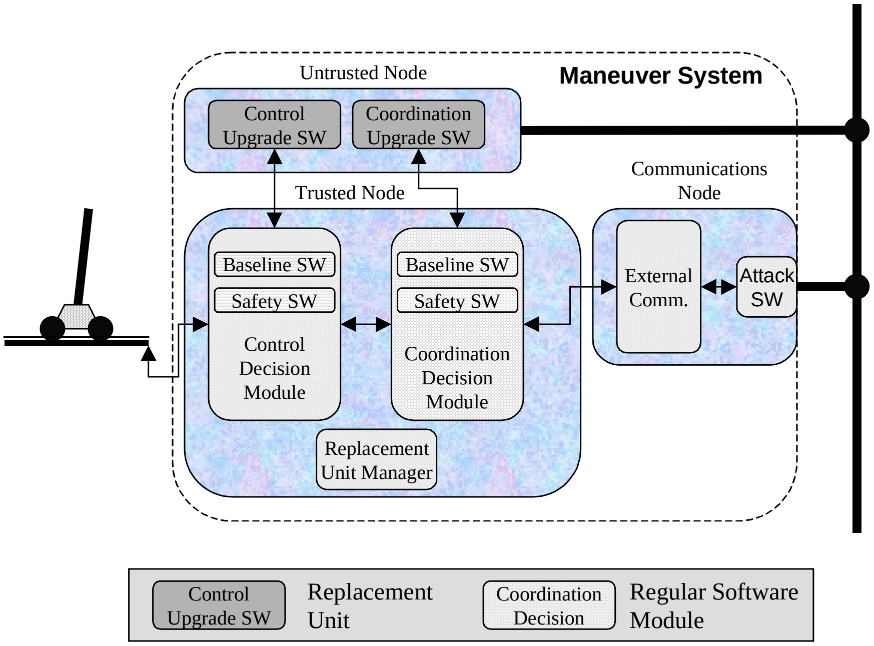


Figure 15: Replacement Units

4.4.4 Graphical User Interface

The Coordinated Prototype GUI started with a design similar to previous prototype GUIs, providing a single point for the entry of commands and viewing of status. Use of the GUI was optional in previous prototypes and failure of the interface would not affect the safe operation of the system. Among the commands required were several related to placing the prototype system into coordinated movement mode and moving the pendulums to designated targets. It was realized that the interface now performed a critical function, with safety and timeliness requirements. The GUI design was modified for greater reliability and fault resistance; the implementation in Java was relatively straightforward.

5 Demonstration Scenarios

The development of the Coordinated Prototype was used to confirm the applicability of the Simplex engineering framework to a communications dependent system. The finished prototype was used as a demonstration vehicle for Simplex technology. One of several systems used to demonstrate Simplex, demonstrations tend to focus on the unique features of the prototype. In this section, the most common prototype demonstrations are briefly described.

5.1 Upgrade Protection

In the upgrade demonstration, a new pendulum controller is introduced which deliberately attempts to make the pendulum loose balance and fall. The defective controller is placed in charge as the leader and permitted to run.

Before the pendulum can fall due to erroneous control inputs, the decision and safety module switches control from the buggy controller to the next controller in the hierarchy of available controllers

5.2 Denial of Communications

With the prototype running, the network connection for one of the maneuver systems is disconnected and left unconnected.

With the pendulum maneuver systems unable to exchange position information, the maneuver systems activate the baseline coordinator and move the pendulums to the last target position received by both systems. The pendulums are then held stationary.

When the network connection is restored, the maneuver systems reestablish communications and again begin to move the pendulums in a coordinated way.

5.3 Message Blocking Attack

Using the communications attack software, the flow of messages between the maneuver systems is reduced.

The pendulum maneuver systems responds by activating the baseline coordinator and reducing movement rates to match the rate of position updates received. When the communications rate increases, the pendulum movement rate is increased.

5.4 Message Bombardment Attack

Using the communications attack software, spurious messages bombard the maneuver systems from the attacker.

The pendulum maneuver systems responds by reducing movement rates to match the rate of position updates received, halting the pendulum if necessary.

6 Analytical Studies

The Coordinated Prototype has been used as a basis for additional analytical studies. These "side studies" used the operational prototype along with development team expertise to apply advanced software engineering techniques to an existing system of tractable size and complexity. For more complete information, consult the references provided for each of the studies.

6.1 Model Based Verification

The fault response capabilities of the Coordinated Prototype were modeled using the Symbolic Model Verifier (SMV) model-checking tool. The SMV is a model checking tool that accepts a finite state representation of a system and a set of properties or claims made about that system in Computational Tree Logic (CTL). The SMV tool checks whether these properties hold for the model. In most cases, it also provides counterexamples for the properties that do not hold.

The case study modeled the outer control loop responsible for coordinating the two pendulum's motion. Detailed information is available in the SEI Technical Report CMU/SEI-98-TR-013 [Srinivasan 98].

6.2 Model Case Study

A separate modeling effort, focusing on the application semantics and the semantics of analytically redundant components is also under way. The study focuses on architectural models and analysis of the configuration consistency using the prototype as the model basis.

In particular, an architectural description language supporting modeling of real-time systems has been extended to allow capture of application semantics and time sensitive properties that typically cause hidden side effects. Design time analysis of system models:

- discovers hidden side effects by detecting inconsistencies in application semantics and schedulability
- identifies inconsistent combinations of component variants, for which the developer can add runtime monitoring
- determines the impact of a change, supporting the developer in the consistent propagation of a change

Detailed information is available in the works by Feiler and Li [Feiler 98] and [Li 99].

7 Lessons Learned

The prototype fulfilled its original goals: the system holds real-time control of the inverted pendulums in the face of communications flaws and faulty software. Coordinated motion is maintained. Software can be upgraded and tested online, as well as in a conventional fashion prior to the system start up.

In creating the prototype, we used a design that split the control and coordination task into two separate sets of analytically redundant components operating in concert within a single system. The use of separate control and coordination loops allows for the separate development of software, more flexibility in scheduling, and the reuse of existing software components. Given the deliberately unreliable communications and dependence on exchanging coordination information between systems, special attention must be paid to ensuring that state information inconsistencies are detected and compensated, but the problem is tractable with careful design of the decision software.

The prototype saw the first application of Lyapunov stability theory and Linear Matrix Inequality (LMI) methodology to the development of Simplex safety controllers (see Appendix C). These techniques provided a mathematical basis for confirming the safety region of the controller and deriving the safety control laws. Their use was helpful in confirming that the existing safety controllers were correct and in developing new software.

Of course the Coordinated Prototype is not fully comparable to an operational environment. We are convinced however, that the design of the prototype is applicable to other systems that face similar requirements.

References/Bibliography

[Avizienis 85] Avizienes, A. “The N-Version Approach to Fault Tolerant Software,” IEEE Transactions on Software Engineering, SE-11(12):1491-1501, December 1985.

[Avizienis 95] Avizienis, A. “The Methodology of N-version Programming”, *Software Fault Tolerance*, John Wiley & Sons, 1995.

[Boyd ND] Boyd, S.; Ghaoul; L. E., Feron, E.; & Balakrishnan, V. *Linear Matrix Inequality in Systems and Control Theory.* SIAM Studies in Applied Mathematics.

[DoD ND] Department of Defense *Joint Vision 2010*.

[Feiler 98] Feiler, P. H. & Li, Jun. “Managing Inconsistency in Reconfigurable Systems,” 145, 5: 172-179. *IEEE Proceedings-Software*, Oct. 1998.

[Gray 90] Gray, J. “A Census of Tandem System Availability Between 1985 and 1990.” *IEEE Transactions on Reliability* 39, 4 (October 1990): 409-418.

[Klein 93] Klein, M. H.; Ralya, T.; Pollak, B.; Obenza, R.; & Harbour, M. G. *A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate-Monotonic Analysis for Real-Time Systems*. Kluser Academic Publishers, 1993. ISBN 0-7923-9361-9.

[Knight 86] Knight, J.C. & Leveson, N. G. “An Experimental Evaluation of the Assumption of Independence in Multi-version Programming.” *IEEE Transactions on Software Engineering*, SE-12(1) (Jan., 1986): 96-109.

[Li 99] Li, Jun & Feiler, P. H. “Impact Analysis in Real-Time Control Systems,” *Proceedings of International Conference on Software Maintenance*, Aug. 1999.

[Rajkumar 95] Rajkumar, R.; Gagliardi, M.; & Sha, L. “The Real-Time Publisher/Subscriber IPC Model for Distributed Real-Time Systems: Design and Implementation,” *The Proceedings of the 1st IEEE Real-Time Technology and Applications Symposium*, May 1995.

[Randell 75] Randell, B. “System Structure for Software Fault Tolerance,” IEEE Transactions on Software Engineering, SE-1: 220-232, June 1975.

[Randell 95] Randell, B. & Xu, J. “The Evolution of the Recovery Block Concept,” *Software Fault Tolerance*, John Wiley & Sons, 1995.

[Seto 99a] Seto, D. & Sha, L. *A Case Study on the Analytical Analysis of the Inverted Pendulum Real-Time Control System* (CMU/SEI-99-TR-023). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999.

[Seto 99b] Seto, D. & Ferreira, E. *A Case Study on the Development of a Baseline Controller for Automatic Landing of an F-16 Aircraft using LMIs* (CMU/SEI-99-TR-020). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999.

[Seto 99c] Seto, D. & Sha, L. *Engineering Method for Safety Region Development* (CMU/SEI-99-TR-018). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999.

[Sha 90] Sha, L. & Goodenough, J. B. “Real-Time Scheduling Theory and Ada.” *IEEE Computer*, April 1990: 53-62.

[Sha 94] Sha, L.; Rajkumar, R.; & Sathaye, S. “Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real-time Systems,” *Proceedings of the IEEE*, January 1994.

[Sha 95] Sha, L.; Gagliardi, M.; & Rajkumar, R. “Analytic Redundancy: A Foundation for Evolvable Dependable Systems,” *The Proceedings of the 2nd ISSAT International Conference on Reliability and Quality of Design*, March 1995.

[Sha 96] Sha, L.; Rajkumar, R.; & Gagliardi, M. “Evolving Dependable Real-Time Systems.” vol. 1, 1335-346. *Proceedings of the 1996 IEEE Aerospace Applications Conference*, New York, N.Y., February 3-10, 1996.

[Srinivasan 98] Srinivasan, G. & Gluch, D. *A Study of Practice Issues in Model-Based Verification Using the Symbolic Model Verifier (SMV)* (CMU/SEI-98-TR-013), ADA 358751. Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1998.

[Yeh 96] Yeh, Y. C. “Triple-Triple Redundant 777 Primary Flight Computer.” vol. 1, 293-307. *Proceedings of the 1996 IEEE Aerospace Applications Conference*, New York, N.Y., February 3-10, 1996.

Appendix A: Communications Attack Modes

The Coordinated Prototype implements communications attacks on inter-system communications links. Direct attacks are limited to the maneuver systems, while the command system is affected only by network congestion when the attacker floods the network with messages. Node to node communications within the maneuver systems utilizes direct serial connections and is not subject to direct attack.

The prototype communications attacks are denial of service attacks where legitimate messages are lost or delayed during transmission. Deceptive jamming attacks, where the contents of messages are altered or syntactically correct but false messages are created, are not allowed.

Inter-system communication is performed on a broadcast media (Ethernet) using UDP/IP protocol.

The user may initiate and control communications attack from the GUI on the command system or by using a command line interface on the attacker.

A.1 Message Blocking

In a message blocking attack legitimate messages are lost during transmission.

Message blocking attacks are implemented by adding agent software to the maneuver system's UDP/IP message routines. The agent software receives commands to set up, start and end communications attacks. The set up command (a “jamming specifier”) sent from the attacker via the network is used to reject a specified percentage of messages directed to or from a specific socket. As individual messages are transmitted or received, a random number is computed and the message is either rejected (lost) or delivered. No history is kept, and following messages have exactly the same chance of passing or failing the check.

A jamming specifier includes the following parameters:

- **Hosts affected**: Set the target for message blocking. Can affect messages directed to the secure node of the maneuver system, upgrade node of the maneuver system, or all nodes of both maneuver systems?
- **Ports affected**: specified port number or all ports on a system
- **Type of message affected**: Receiving (incoming), sending (outgoing), or both
- **Chance of loss**: The percentage chance that a message arriving or leaving the port will be lost, ranging from 0 to 100%
- **Blocking on duration**: length of time a jamming specifier will be active, 0-99 seconds
- **Blocking off duration**: length of time a jamming specifier will sleep before the next activation, with a range of 0-99 seconds
- **Number of blocking events**: The numbers of times an attack will be peformed, ranging from 0-99 with 0 standing for continous exectuion

Multiple jamming specifiers may be set on any socket. The agent software scans the list of jamming specifiers and applies the first active specifier, which matches the search criteria. Any additional specifiers in the list are ignored.

A.2 Message Bombardment

In a message bombardment (or “packet barrage”) attack spurious messages are generated and directed at target communications links.

Message bombardments use ICMP echo packets to create traffic on the network. ICMP echo packets are normally used to establish the presence of active nodes on a network (ping facility); consequently receipt of an ICMP causes the IP stack to return an echo reply packet. The attacker system can direct the bombardment at a specific node using the directed ICMP packets or attack all nodes using broadcast ICMP packets.

The effect of message bombardment attacks varies according to network topology. When the network uses a common cable between all nodes[4], the return packets affects all nodes. Since a single broadcast ICMP echo packet generates a response from each node, the amount of network traffic depends on the number of nodes. If a network switch is used, the return packets affect only the nodes connected to that switched segment. In the prototype, a network switch with one node per segment was normally used.

Message bombardment is controlled by the following parameters:

- **Level**: The level of attack, expressed as a percentage, ranging from 0-100
- **Duration**: length of time the attack will be active, 0-99 seconds

[4] Passive hubs are equivalent to straight cabling between nodes.

- **Gap**: length of time between the end of an attack and the next bombardment, with a range of 0-99 seconds
- **Host**: System to receive a jamming attack, one of Maneuver System 1, Manuever System 2 or all systems

Appendix B: Reliability Analysis

In this section, we provide a simple analysis of the reliability and availability of analytically redundant control systems and compare it with N-Version programming and Recovery Blocks [Avizienis 95 and Randell 95]. First, we consider N-Version programming. Supposed that 3 versions of high performance controllers are used and the faults are independent. Suppose that each version has the same reliability *r(t)*. The system reliability of this system is $RN(t) = r(t)^3 + 3r(t)^2(1 - r(t))$.

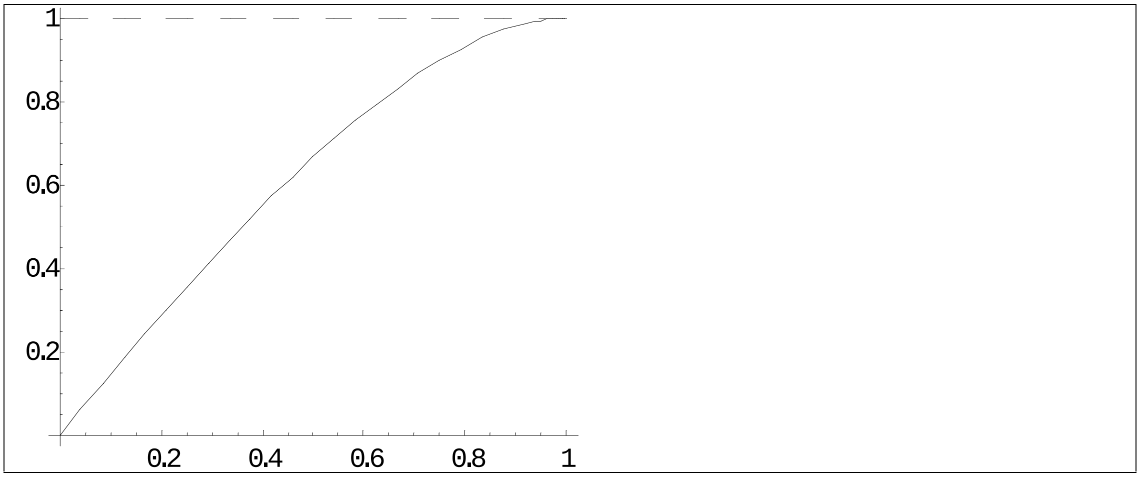


Figure 16: Plot of High Performance Controller Reliability

To compare N-Version programming with an analytically redundant control system (Simplex architecture), let’s make the minimal change. We replace one of the three high performance controllers with a simple high assurance controller with reliability rH(t). Under the Simplex architecture, the two high performance controllers are configured as primary and standby. There are two figures of interest. First, the reliability of the high assurance control software is just rH(t) and we assume that rH(t) >> r(t). Under Simplex architecture, the high performance control system’s reliability is $RH(t) = (r(t)^2 + 2r(t)(1 - r(t)))\ rH(t)$. First, note that when $rH(t) = 1$, $RH(t) - RN(t) = r(t)(1 - r(t))^2 \geq 0$. That is, the reliability of the high performance control under Simplex architecture is higher than that under 3-version programming, provided that the high assurance control has perfect reliability.

When the high assurance control reliability is not perfect (i.e., $rH(t) < 1$) the high performance controller’s reliability under Simplex is still higher if the high assurance controller has a very high relative reliability (i.e., $RH(t) \geq RN(t)$ *if* $rH(t) \geq (RN(t) / (r(t)^2 + 2r(t)(1 - r(t))$ and vice versa. Figure 16 plots that for a given value of high performance controller reliability, *r(t)*, the value of the high assurance controller reliability *rH(t)* at which the reliability of the high performance control under Simplex and 3-version programming are equal (i.e., *RH(t) = RN(t))*.

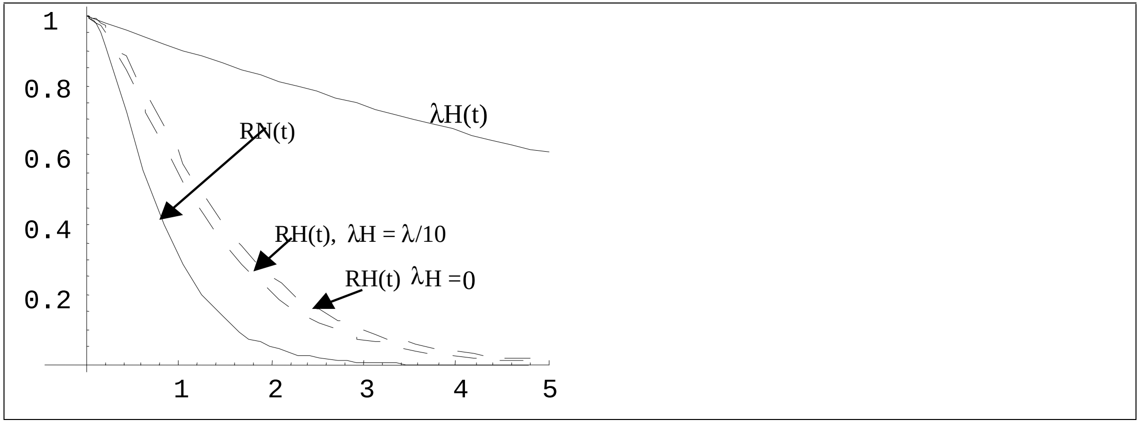


Figure 17: Plot of High Performance Control Reliability Under 3-Version Programming

Simplex architecture provides not only a high degree of reliability with respect to stability and controllability but also a higher reliability in high performance control, provided that the reliability of the high assurance controller is much higher than the high performance control (i.e., *rH(t) >> r(t))*. Figure 17 is a plot of the high performance control reliability under 3-version programming, *RN(t),* the reliability of the high assurance control, rH(t) when the high assurance controller has a failure rate λH equals to 1/10 of the failure rate of high performance controller, λ. In addition, the two dashed lines plot the high performance control reliability under Simplex when 1) λH = 0 and when λH = λ/10. The horizontal axis is λ from 1 to 5 and the duration of mission is normalized to 1.

High reliability translates into longer mean time to failure, *MTTF,* since *MTTF* is the integration of reliability over time. When *RH(t) >> r(t),* the *MTTF* of high performance control under simplex is longer than that under 3-version programming. Next, the mean time to repair *MTTR* under Simplex is also shorter since the high performance system can be restarted quickly when the vehicle is able to maintain its stability and controllability under high assurance control. When the high performance control system fails under 3-version programming, the vehicle loses computer control completely and could become unstable. The “repair time” to get back to normal control could take much longer. Since availability is equal to *MTTF/(MTTF + MTTR*), it follows that the availability of high performance control is higher under the assumption of *RH(t) >> r(t)*. Note that in the analysis above, the N-Version programming idea was incorporated into the Simplex architecture with the use of two different versions of high performance controllers.

Finally, we want to comment on the similarities and differences between the Simplex architecture and recovery blocks from the viewpoint of reliability model structure. Let us configure the two high performance controllers and the high assurance controller in the form of recovery blocks. The system reliability is $RB(t) = (1 - (1 - r(t))^2(1 - rH(t))\ rA(t)$*, where* $rA(t)$ is the reliability of the acceptance test. Under the assumption of a perfect acceptance test, rA(t) = 1, recovery blocks have a higher reliability, that is $RB(t) - rH(t) = r(t)^2\ (1 - rH(t)) \geq 0$. Intuitively, this follows from the fact that under Simplex if the high assurance controller fails the control system fails. However, the control system would not fail as long as one high performance controller still works, even though the high assurance controller fails. It follows that the reliability of the high performance controller is also higher under recovery blocks, since as long as any one of the high performance controller works, the system will be under the control of the working high performance controller, even if the high assurance controller fails, provided that the acceptance test is perfect. The effect of a near perfect acceptance test can also be carried out in a way similar to what we have done for the Simplex architecture by replacing $rH(t)$ with $rA(t)$. The challenge is, however, to write an effective acceptance test.

Appendix C: Application of Lyapunov Stability Theory to Safety Controller Design

As one of the critical contributions of the Simplex architecture, tolerance of semantic faults is accomplished by implementing a safety controller, which runs in parallel with the experimental controller and will take over the control of the physical system once a fault is detected. The success of such a fault tolerant mechanism is highly dependent on the detection of faults and the design of the safety controller. Lyapunov stability theory provides a foundation for systematically deriving a criterion in semantic fault detection and developing a safety controller by formulating a linear matrix inequality (LMI) problem, which can be solved by the newly developed interior-point optimization algorithms.

A semantic fault can be detected by monitoring the dynamics of the physical system, which is represented by the state of the system, $x(t)$. In the Simplex architecture, a semantic fault is defined as causing the physical system to fail (i.e., not being able to carry out its normal operation and eventually failing). To be more precise, we say that a semantic fault will lead the physical system to an undesired state where no available control can bring it back to normal operation. In most control systems, the fundamental control issue is to maintain the stability of the system, and therefore, a state from which no available control can keep the system stable should by all means be avoided. With respect to stability, a semantic fault is defined as driving the physical system to instability. To check if a semantic fault occurred during the operation of the experimental controller, one needs to evaluate the dynamic behavior of the physical system to see if it will become unstable. We say that the physical system is safe at time t if it is in a state $x(t)$ from which the available control authority can keep it stable in the future. As the safety controller will provide protection against semantic faults in the Simplex architecture, the physical system is required to always be in one of the states from which the safety controller is able to maintain the stability of the system. In terms of Lyapunov stability theory, the collection of these states is called the stability region of the safety controller. Hence, we conclude that the stability region of the safety controller serves as a criterion in semantic fault detection, namely, a semantic fault has occurred if the state of the physical system is out of the stability region of the safety controller. Moreover, the stability region of the safety controller will also be the recoverable region in the Simplex architecture. It should be noted, however, such a recoverable region is only useful when the stability region of the safety controller is large enough to cover all the trajectories of the physical system in normal operation. This issue will be addressed in design of the safety controller.

The stability region of a safety controller can be characterized by a Lyapunov function. In control systems where stability is a concern, an equilibrium or equilibria can always be found as special states of the system, states at which the system will stay forever once it reaches one of them. In the safety control, we are interested in the so-called asymptotically stable equilibrium, to which the system will converge asymptotically if the system starts close to the equilibrium. Therefore, a stability region is a neighborhood of the equilibrium, from which the system trajectories will eventually converge to the equilibrium state. Formally, we express such region as a set of system states given by:

$$S = \{x : V(x) \leq 1, \; V(x_e) < 1\}$$

with x_e the equilibrium state and function $V(x)$ satisfying
the following conditions:

1). x_e is the unique minimum of $V(x)$ in S;

2). The time derivative $\dot{V}(x) < 0, \; \forall x \neq x_e \in S$.

Function $V(x)$ is then referred to as a Lyapunov function. As illustrated above, the existence of a Lyapunov function determines the dynamics of the physical system, namely, the decrease of the value of the Lyapunov function evaluated along the system trajectory restricts the trajectory such that it will converge to x_e asymptotically. During the course of reaching x_e, the trajectory will be bounded by a closed boundary x_b given by $V(x_b) = V(x_0)$ with x_0 the initial state. By thinking of a Lyapunov function as the energy of the system, the dynamics of the system undergoes an energy dissipation process with the minimum energy at the equilibrium point. To summarize, we conclude that the stability region of a safety control can be expressed as a set $S = \{x : V(x) \leq 1\}$ with $V(x)$ a Lyapunov function and $x_e \in S$.

To construct a Lyapunov function to represent the stability region, we apply Lyapunov stability theory. For the sake of developing a systematic approach, we assume the physical system is linear time invariant (LTI). While this may not be true in practice, most of the physical systems can be linearized about the equilibrium state and their dynamics can be approximately described by LTI systems. For a given LTI system, Lyapunov stability theory states: the system is asymptotically stable at the equilibrium state x_e if and only if there exists a quadratic function of the state variables $V(x) = (x - x_e)^T P(x - x_e)$ with P a positive definite matrix, that has negative derivatives. This result narrows the search for a Lyapunov function to finding a positive definite matrix P. As we mentioned earlier, the stability region will be used as the recoverable region in the Simplex architecture, and hence, an additional requirement needs to be imposed in solving for matrix P, that is, the stability region should be as large as possible. Summarizing all the requirements, we post two optimization problems for finding the largest stability region:

1) For a given safety controller $u = Kx$ (e.g., a controller that has been used extensively in the past with an approved record of reliability), find matrix P such that the size of the stability region $S = \{x : (x - x_e)^T P(x - x_e) \leq 1\}$ is maximized subject to state and control constraints.

2) When the safety controller is to be designed, let it be a linear state feedback control $u = Kx$ with K being the control gain to be determined. Find the matrix P and the control gain K such that the size of the stability region $S = \{x : (x - x_e)^T P(x - x_e) \leq 1\}$ is maximized subject to state and control constraints.

Both of these two problems can be formulated as LMI problems and solved by using the newly developed interior-point optimization approaches. Detailed procedures for deriving the solutions will be reported elsewhere. In either case, we obtain a safety controller $u = Kx$ with the largest stability region, or the recoverable region in the Simplex architecture, given by the set $S = \{x : (x - x_e)^T P(x - x_e) \leq 1\}$.

Implementation of the safety controller and the recoverable region described above is rather trivial. With the sampled data $x(t)$ in each sampling period, the safety control command is simply the multiplication of the control gain and the state data. The check for semantic faults is carried out by computing the quadratic function $(x - x_e)^T P(x - x_e)$ to see if it is greater than constant 1. If this is the case, then a semantic fault has occurred.

In summary, we conclude that, for a LTI system, the recoverable region or semantic fault detection can be systematically derived by establishing a stability region for the safety controller, which can be a given controller or a controller to be designed. The resulting recoverable region will be the largest with respect to the given safety controller, or the largest in all possible linear feedback controllers if the safety controller is designed.

Index